ALLA SCOPERTA DEL VIC 20

Rita Bonelli
Daria Gianni

architettura e tecniche di programmazione



GRUPPO EDITORIALE JACKSON

ALLA SCOPERTA DEL VIC 20

architettura e tecniche di programmazione

Rita Bonelli
Daria Gianni



GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

Il Gruppo Editoriale Jackson ringrazia per il prezioso lavoro svolto nella stesura dell'edizione italiana la signora Francesca Di Fiore, e l'Ing. Roberto Pancaldi.

Fotocomposizione: Lineacomp S.r.l. - Via Rosellini, 12 - 20124 Milano

Stampato in Italia da:
Tipo Lito Ferrari Cesare & C. — Clusone (BG)

SOMMARIO

APPENDICI

RINGRAZIAMENTI

Le autrici ringraziano James Bachmann, Amministratore Delegato, e Sergio Messa, Direttore Generale della Commodore Italiana s.r.l., per avere messo a disposizione le apparecchiature e la documentazione necessarie alla realizzazione dell'opera.

Ringraziano i giovani Fabio Racchi, Luciano Pazzucconi e Giovanni Valerio, per l'aiuto fornito nella messa a punto di alcuni programmi.

Tutti i programmi che compaiono nel testo sono stati provati sul calcolatore; i numeri esadecimali, salvo avviso contrario, sono contraddistinti da una H finale.

PREFAZIONE

Questo libro esce un pò in ritardo rispetto alle promesse fatte ai lettori con il volume "Vogliamo Incominciare Così"; me ne scuso, ma in mezzo ci sono stati tanti altri lavori. Nel frattempo sono stati venduti molti VIC 20, molte persone si sono avvicinate al mondo dell'informatica per mezzo di questo calcolatore, e probabilmente un buon numero di queste ci hanno preso gusto.

Il libro si propone di soddisfare diverse esigenze; i capitoli che trattano i file su disco e cassetta, la stampante e alcuni cartridge dovrebbero rappresentare una esauriente integrazione per coloro che, con l'aiuto del primo volume, sono diventati "utenti BASIC" del VIC. Per chi invece desidera approfondire anche l'aspetto hardware, per arrivare a un tipo di programmazione più sofisticata, presenteranno un maggior interesse i capitoli sulle porte di I/O, sul chip di interfaccia video (VIC 6561) e sul linguaggio macchina del calcolatore.

Da tutto questo, insieme con gli argomenti trattati nel primo volume, che qui si è cercato di non ripetere, esce una panoramica abbastanza vasta e completa del VIC.

Vorrei sottolineare un fatto che mi sembra importante. Oggi si parla molto di informatica, di seconda alfabetizzazione. E' vero, sembrerà strano in futuro, sembra già quasi strano oggi, non sapere usare il calcolatore. Ma per imparare ad "usare" il calcolatore bisogna ... "usarlo". Non si deve aver paura di metterci le mani sopra; se si ha un dubbio, ci si arma di pazienza, si prova e ce lo si toglie. Non è necessario disporre di apparecchiature costose; basta un personal e il desiderio di approfondire le cose.
Mi auguro che i lettori di questi libri abbiano imparato, oltre che a lavorare con il VIC, anche questa "filosofia", che senz'altro li aiuterà a lavorare con qualsiasi altro calcolatore.

Il presente volume è il frutto dell'impegno di due persone; spero che il lavoro mio e di Daria Gianni non deluda coloro che l'attendevano.

(Rita Bonelli)

CAPITOLO 1

MICROPROCESSORE 6502

1.1 ARCHITETTURA DEL SISTEMA VIC 20

La figura seguente mostra lo schema tipico di un sistema a microprocessore, come è il VIC 20.

Figura 1.1 Schema di un sistema a microprocessore

Gli elementi fondamentali del sistema sono:
.Unità Centrale (CPU)
.Memoria a sola lettura (ROM)
.Memoria a lettura e scrittura (RAM)
.Interfacce per i dispositivi di I/O.

Tra i due termini microprocessore e microcalcolatore esiste una differenza. Il microprocessore è la CPU del sistema; è in genere un unico integrato. Esso, insieme con le memorie, le interfacce di I/O, almeno un dispositivo di ingresso (tastiera) e almeno un dispositivo di uscita (stampante o video), costituisce un microcalcolato-

re. In sostanza un microcalcolatore è un calcolatore elettronico che ha come CPU un microprocessore. Il VIC è un microcalcolatore che ha come CPU il microprocessore 6502.

Le connessioni tra i vari componenti sono realizzate tramite bus: per bus si intende un insieme di linee, ognuna delle quali trasporta 1 bit, lungo le quali fluiscono le informazioni da una o più sorgenti a una o più destinazioni. In sostanza un bus è un canale unico che collega diversi dispositivi; le comunicazioni possono avvenire naturalmente solo tra due dispositivi alla volta, e quasi sempre uno dei due è la CPU.

I bus del sistema VIC 20 sono 3:

.il bus degli indirizzi, di 16 linee, su cui transitano gli indirizzi generati dalla CPU per comunicare con una parola di memoria (byte) o con un dispositivo di I/O;

.il bus dei dati, di 8 linee, su cui transitano i dati dalla CPU verso le memorie e i dispositivi di I/O, e viceversa;

.il bus di controllo, di 7 linee, su cui transitano i segnali di controllo che guidano le diverse operazioni del sistema nell'esecuzione del programma.

La CPU è costituita dal microprocessore 6502, a cui è dedicato il paragrafo seguente.

La ROM e la RAM costituiscono la memoria complessiva su cui opera il calcolatore; la prima contiene quei programmi del sistema operativo (MONITOR) che devono mettere il calcolatore in condizione di funzionare al momento dell'accensione, e che quindi non possono venire persi quando il calcolatore si spegne. La RAM, invece, contiene i programmi e i dati che vengono di volta in volta scritti ed usati dall'utente; poichè la RAM è volatile, questi vengono persi quando si spegne il calcolatore, ma possono essere conservati su memorie di massa come nastri e floppy-disk. Si noti per inciso che questi ultimi dispositivi, pur rappresentando

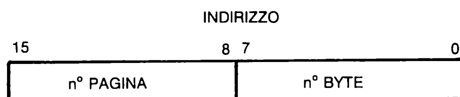


Figura 1.2 Schema dei 2 byte che formano un indirizzo

come si è detto memorie di massa o ausiliarie, vengono nel seguito trattati come dispositivi di I/O.

Poichè il bus degli indirizzi è di 16 linee, la CPU può generare fino a "2 elevato a 16" (65536) indirizzi diversi; l'insieme della ROM e della RAM potrebbero avere un totale di 64K byte di capacità. Va però considerato che il 6502 usa per le comunicazioni con i dispositivi di I/O, la tecnica che va sotto il nome di "I/O memory mapped" (o, con un brutto italiano, I/O mappato in memoria); ciò significa che ad ogni dispositivo, o meglio, ad ogni interfaccia, vengono associati uno o più indirizzi, a seconda del numero dei registri programmabili dell'interfaccia stessa, con i

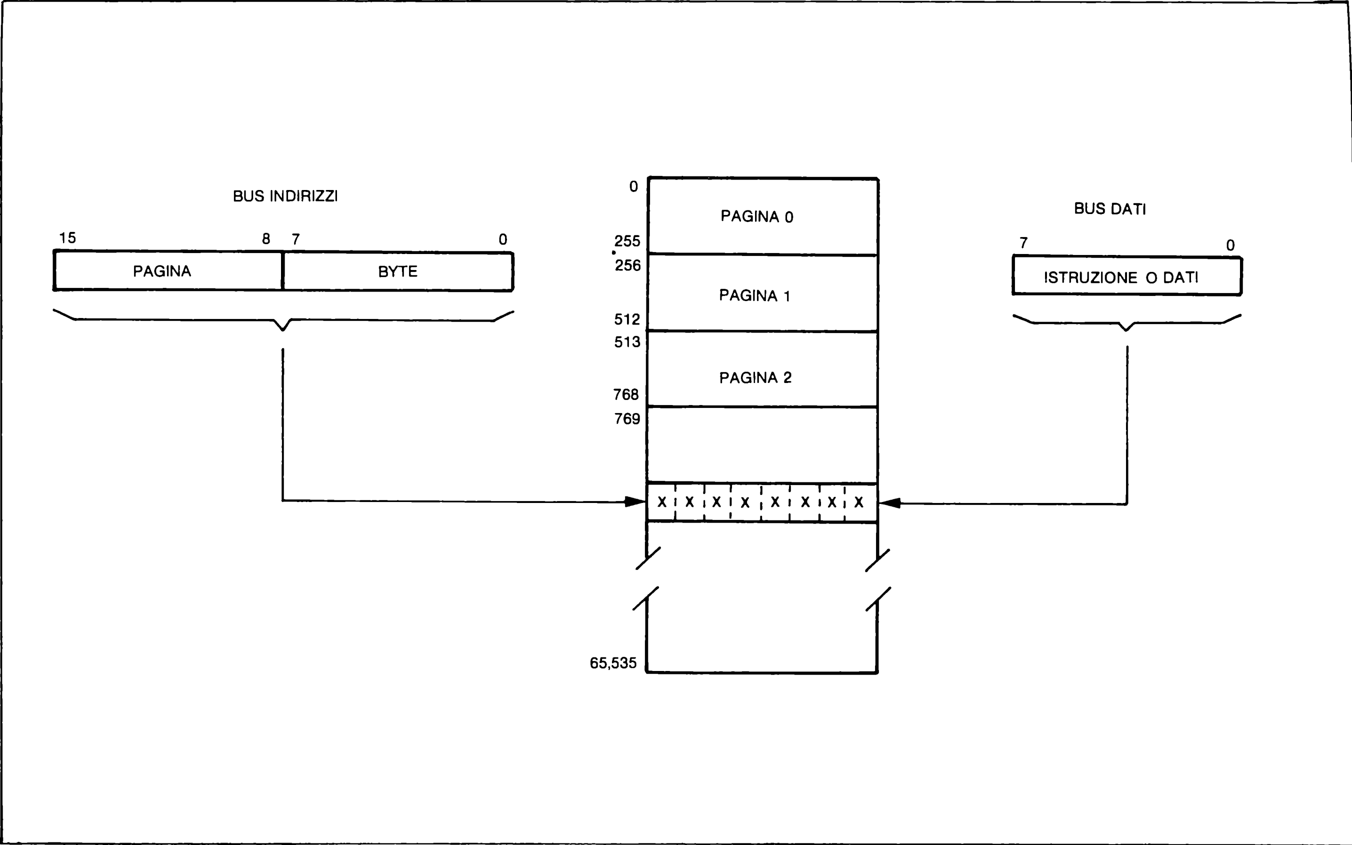


Figura 1.3 Relazioni tra indirizzi, locazioni e contenuti di memoria

quali essa viene individuata e selezionata per le comunicazioni. I segnali di controllo per le operazioni di I/O sono gli stessi di quelli per le operazioni in memoria. Questi indirizzi, non possono naturalmente corrispondere a locazioni di memoria, per non avere ambiguità nell'indirizzamento. Una parte degli indirizzi che la CPU può generare viene quindi destinata ai dispositivi di I/O, a scapito della capacità di memoria. Va detto però, che sul VIC questa riduzione è quasi insignificante; infatti gli indirizzi dedicati ai dispositivi di I/O occupano meno di 4K, e precisamente da 37136 a 37887 (9110H-93FFH) e da 38912 a 40959 (9800H-9FFFH).
Tornando alle memorie, si fa notare che può essere utile considerare gli indirizzi come costituiti da due parti di 8 bit ciascuna, come nella figura 1.2.
In questo modo si può immaginare la memoria come suddivisa in 256 pagine di 256 byte ognuna; il byte più significativo (HIGH) dell'indirizzo contiene il numero della pagina, quello meno significativo (LOW) il numero del byte all'interno di quella pagina.
La relazione tra indirizzi, locazioni di memoria e loro contenuti è schematizzata nella figura 1.3.
Due parole, infine, per chiarire la differenza che c'è tra un dispositivo di I/O e un'interfaccia di I/O. I dispositivi di I/O sono quelli ben noti all'utente, che li usa per comunicare con il calcolatore: tastiere, video, stampanti, registratori a cassette, unità a dischi e floppy disk, per citare i più comuni. I linguaggi, se così si può dire, di questi dispositivi, in termini di segnali elettrici, di codici, di funzionamento non sono in generale compatibili con quello del calcolatore; si usano allora circuiti particolari, detti appunto di interfaccia, che non solo consentono il collegamento e la comunicazione, ma presentano anche una notevole flessibilità e facilitano la gestione delle comunicazioni stesse. Quasi tutti questi dispositivi sono programmabili (in questo consiste la loro flessibilità), cioè posseggono al loro interno dei registri, accessibili tramite degli indirizzi, come parole di memoria, che possono essere programmati in modo da ottenere un certo funzionamento dell'interfaccia.
Le due principali interfacce di I/O del VIC sono quella per la gestione del video (chip 6561) e le porte per la trasmissione dei dati (chip 6522) a cui sono dedicati due capitoli del libro.

1.2 SCHEMA A BLOCCHI FUNZIONALE E PIN-OUT DEL 6502

L'Unità Centrale (CPU: Central Processing Unit) del VIC 20 è il microprocessore 6502, un unico circuito integrato (chip). Nella figura 1.4 è mostrato lo schema a blocchi funzionale della struttura interna del 6502.

Nella figura si possono individuare due sezioni: una di controllo a destra e una dei registri a sinistra.
Nella sezione di controllo si vedono i seguenti blocchi:

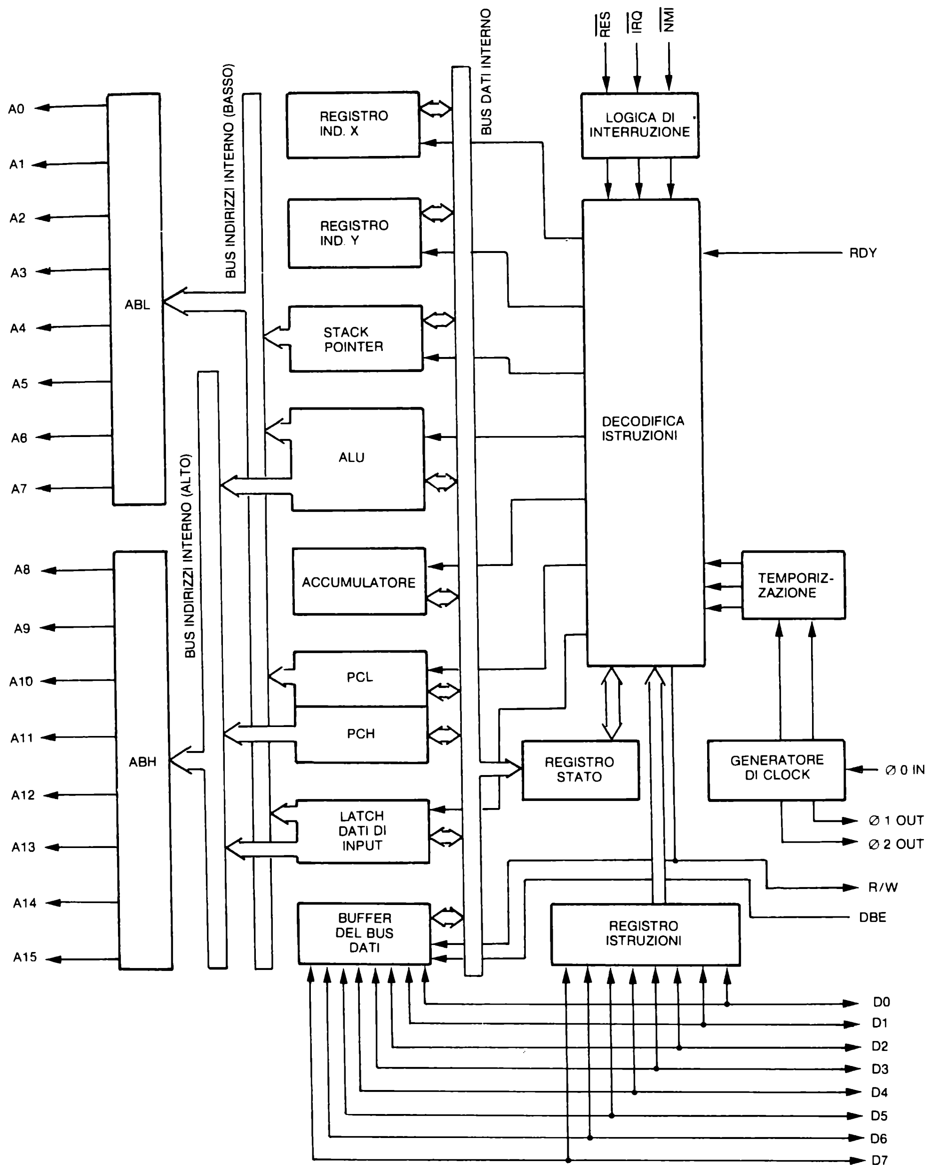


Figura 1.4 Schema funzionale 6502

Logica di decodifica istruzioni: sono i circuiti che eseguono la fase di decodifica dell'istruzione. Si ricorda che l'esecuzione di ogni istruzione (in linguaggio macchina) di un programma si articola in 3 fasi fondamentali:

1) fase di fetch (prelievo), in cui l'istruzione, o più precisamente il suo codice operativo, viene prelevato dalla memoria, in cui risiede il programma, e portato nella CPU;

2) fase di decodifica, in cui l'istruzione viene riconosciuta e interpretata dalla CPU, che genera di conseguenza i segnali e le temporizzazioni necessarie per eseguirla;

3) fase di esecuzione, in cui l'istruzione viene propriamente eseguita.

Il programma da eseguire si presenta come una sequenza di istruzioni consecutive nella memoria del calcolatore; ogni istruzione è costituita da un primo byte che

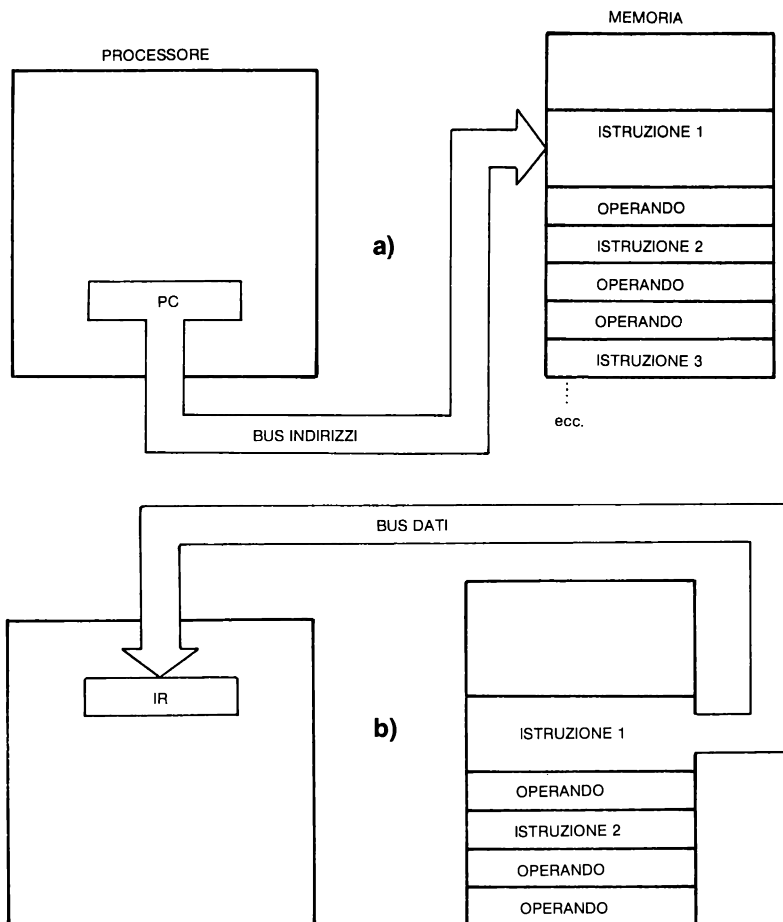


Figura 1.5 a), b) - Esecuzione di una istruzione.

contiene il codice operativo, e da uno o più byte successivi che contengono l'operando. Per operando si intende o il dato su cui deve operare l'istruzione, o le informazioni necessarie a reperire in memoria il dato, o un indirizzo necessario per la prosecuzione del programma. La sequenza delle operazioni del microprocessore per l'esecuzione di un'istruzione è illustrata nella figura 1.5.

La fase di fetch è illustrata nei passi a) e b). Nel passo a) il PC (Program Counter: Contatore di Programma) contiene l'indirizzo della prima locazione dell'istruzione da eseguire, che nell'esempio occupa 2 byte, il primo per il codice operativo e il secondo per l'operando. Tale indirizzo viene caricato sul bus degli indirizzi, e

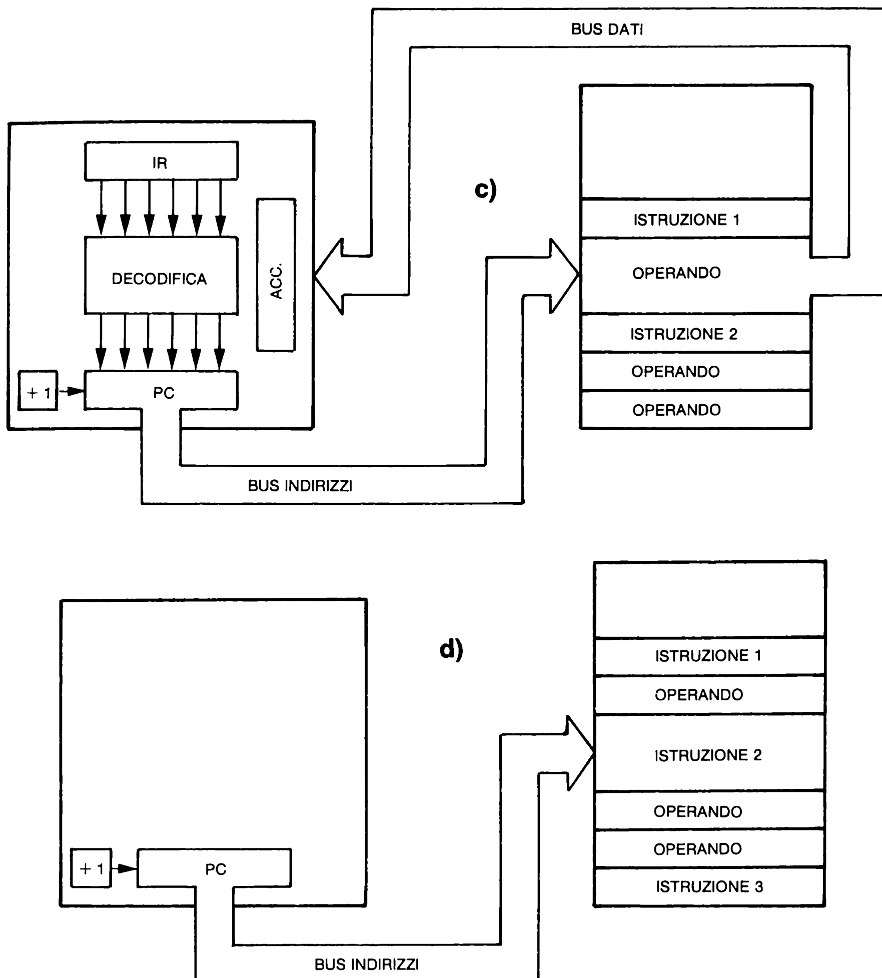


Figura 1.5 c), d) - Esecuzione di una istruzione.

presentato all'ingresso della memoria, insieme con i segnali di controllo che segnalano un'operazione di lettura in memoria da parte della CPU. Nel passo b) la memoria risponde ponendo sul bus dei dati il codice operativo dell'istruzione, che viene caricato nel registro istruzione della CPU. Segue la fase di decodifica, in cui il contenuto del registro istruzioni viene decodificato, il PC viene incrementato di 1, in modo da puntare al successivo byte di memoria, l'operando viene trasferito nell'accumulatore, e, in base alla decodifica la CPU passa alla vera e propria fase di essecuzione (passo c). Nel passo d) il PC viene ulteriormente incrementato, e punta al codice operativo dell'istruzione successiva; dopodichè il ciclo si ripete.

.Logica di interruzione: sono i circuiti che trattano i segnali di interruzione che provengono alla CPU dall'esterno. Sul concetto di interruzione, fondamentale nel funzionamento del calcolatore, si tornerà in seguito.

.Generatore di CLOCK: anche questo concetto è fondamentale nel funzionamento del calcolatore. Tutte le operazioni del 6502 sono "cadenzate" dagli impulsi di un segnale che si presenta come segue:

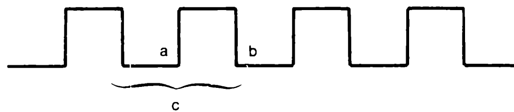


Figura 1.6 Segnale di clock

cioè avvengono in corrispondenza di un fronte positivo (a) o di un fronte negativo (b) di un impulso di questo segnale. Un periodo di questo segnale (c) prende il nome di ciclo di clock; ogni istruzione macchina dura un certo numero di cicli di clock, numero che varia da istruzione a istruzione.
Il dispositivo che genera questo segnale prende il nome di clock (orologio), e il segnale si chiama segnale di clock. Tanto più alta è la frequenza del segnale di clock, tanto più veloce sarà il microprocessore.
Per tornare al blocco in questione, esso rappresenta i circuiti che ricevono in ingresso un segnale di clock generato da un circuito esterno, che temporizza le operazioni del 6502, e ne rendono disponibili due verso i dispositivi esterni.

.Registro di stato: è uno dei registri della CPU, che si è però collocato nella sezione controllo, in quanto è lo strumento di controllo fondamentale della CPU. Esso contiene, rappresentate con lo stato di alcuni suoi bit, le informazioni fondamentali sullo stato della CPU. Questo, come tutti gli altri registri del 6502 accessibili all'utente, sarà descritto in un apposito paragrafo.

.Registro istruzioni: è il registro in cui viene trasferito il codice operativo dell'istruzione prelevata in memoria. Non è accessibile all'utente.

I blocchi principali della sezione registri sono:

.Buffer del bus dati: è il registro che contiene il dato inviato o ricevuto dalla CPU. Non è accessibile all'utente.

.Program Counter (contatore di programma): contiene in ogni istante l'indirizzo dell'istruzione successiva a quella in esecuzione.

.Accumulatore: è un registro di uso generale, su cui operano diverse istruzioni, come si vedrà.

.ALU (Arithmetic Logic Unit: unità aritmetico logica) è la "calcolatrice" interna alla CPU, cioè l'insieme dei circuiti che eseguono i calcoli.

.Stack Pointer (puntatore all'area di stack): contiene l'indirizzo del primo byte libero di una zona di memoria (chiamata area di stack) che viene usata sia dall'utente che dal sistema per conservare temporaneamente dei dati.

.Registri Indice: si usano per realizzare l'indirizzamento indicizzato, tecnica di cui si parlerà in seguito.

Anche i collegamenti interni tra i diversi blocchi della CPU, come quelli visti nell'architettura del microcalcolatore, sono realizzati tramite bus.

Il chip 6502 è dotato di 40 piedini metallici, che rappresentano i contatti verso l'esterno della CPU. I piedini si chiamano "pin" e lo schema del chip in cui sono evidenziati il numero di ogni piedino e la sua funzione prende il nome di pin-out. Il pin-out del 6502 è mostrato nella figura seguente.

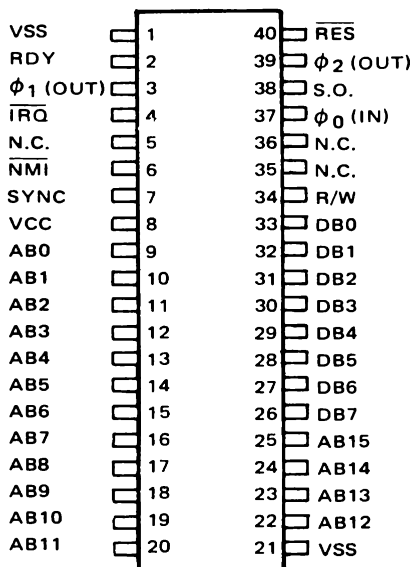


Figura 1.7 PIN-OUT del 6502

Il significato dei vari piedini è il seguente:

D0-D7: linee del bus dei dati

A0-A15: linee del bus indirizzi

Φ0: segnale di clock in ingresso alla CPU

Φ1, Φ2: segnali di clock in uscita dalla CPU

SYNC: uscita che viene attivata in corrispondenza alla fase di fetch

R/W: uscita che segnala l'esecuzione di un' operazione di lettura, quando è attivo, o di scrittura se è disattivato

RDY: ingresso che informa la CPU se la memoria con cui deve comunicare è pronta o meno a ricevere o a trasmettere un dato

IRQ (soprassegnato), NMI (soprassegnato): due linee di ingresso su cui la CPU può ricevere interruzioni da un dispositivo esterno; le interruzioni che arrivano alla prima sono "mascherabili", cioè possono essere ignorate dalla CPU, le altre no

RES (soprassegnato): è un ingressso che inizializza la CPU

S.O. (set overflow): è un ingresso che setta via hardware il flag di overflow del registro di stato

Vcc, Vss: sono i piedini dell'alimentazione (+5V, 0V)

I simboli di alcuni piedini sono soprassegnati con una lineetta; si dice che quei segnali sono attivi bassi, il che significa che hanno effetto quando sono a 0; gli altri, che si dicono attivi alti, hanno effetto quando sono a 1.

1.3 REGISTRI DEL 6502

Nella figura 1.4. sono riportati i registri interni della CPU del VIC; essi sono:

a) registro Program Counter (PC)

b) registro Istruzioni (IR)

c) registro di Stato (P)

d) registro Puntatore area stack (S)

e) registro Accumulatore (A)

f) registri Indice (X e Y)

Il Program Counter è a 16 bit, mentre tutti gli altri sono a 8 bit.
Il PC contiene in ogni istante l'indirizzo dell'istruzione successiva a quella in esecuzione, e viene incrementato automaticamente dalla CPU durante l'esecuzione di un programma. Gli indirizzi di memoria sono a 16 bit: la parte alta dell'indirizzo, che indica il numero della pagina, occupa il byte alto del PC e si indica con PCH (Program Counter High). La parte bassa dell'indirizzo, che contiene il numero del byte all'interno della pagina, occupa il byte basso del PC e si indica con PCL (Program Counter Low).
Il registro Istruzioni contiene il codice operativo dell'istruzione che deve essere eseguita, e lo trasmette alla logica di decodifica che, come si è visto, provvede ad interpretarlo.
Il registro di Stato è a 8 bit; 7 di questi sono utilizzati per segnalare lo stato della CPU, ognuno con un significato particolare, e si indicano con il nome di FLAG (bandiere).
Nella figura seguente è schematizzato il registro di stato; con le lettere si indicano i diversi flag.

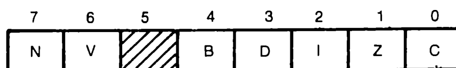


Figura 1.8 FLAG del registro di stato

I FLAG si possono considerare divisi in tre gruppi: al primo appartengono quelli che sono controllati dall'utente, al secondo quelli controllati sia dall'utente che dalla CPU, al terzo quelli controllati esclusivamente dalla CPU.
Il flag D (di modo decimale) è l'unico del primo gruppo: esso è messo a 0 o a 1 dal programma, a seconda che si vogliano eseguire operazioni aritmetiche su numeri rappresentati in binario puro o in BCD. La rappresentazione in binario puro si dà per scontata. In quella in BCD, ogni cifra del numero decimale viene codificata separatamente in 4 bit, che è il minimo necessario per dare 10 codici diversi. Così il numero 53 si presenta in BCD come:

0101	0011
5	3

mentre in binario puro si presenta come: 00110101.
In BCD in ogni byte si possono rappresentare numeri di 2 cifre (4 bit per cifra), da

00 a 99, e naturalmente le regole delle operazioni aritmetiche su numeri in BCD sono diverse da quelle per le stesse operazioni su numeri in binario; basti pensare che in binario si verifica un riporto dal bit 3 al bit 4 quando il risultato di un'operazione ha dato nei primi 4 bit un valore superiore a 15, mentre in BCD tale riporto si verifica per risultati maggiori di 9.
Al secondo gruppo appartengono 3 flag: il Carry, l'Overflow e il flag di disabilitazione delle interruzioni.
Il flag C (Carry=riporto) viene modificato automaticamente dalle CPU in seguito ad alcune operazioni aritmetiche, e segnala in genere, con un 1, un riporto dal bit più significativo del risultato (l'ottavo); inoltre viene usato come nono bit in alcune operazioni di rotazione e di scorrimento, che spostano tutti i bit di un byte verso destra o verso sinistra di una posizione. Infine può essere modificato dal programma con opportune istruzioni.
Il flag di Overflow (V) ha una funzione analoga a quella del Carry per le operazioni su numeri con segno, in cui il bit più significativo è usato appunto per il segno (si veda paragrafo 1.4); segnala in sostanza un riporto dal settimo bit. Inoltre può essere modificato dal programma con opportune istruzioni.
Il flag di disabilitazione delle interruzioni (I) mette in condizione la CPU di accettare o meno interruzioni mascherabili; la CPU stessa lo pone a 1 quando accetta un'interruzione, per evitare di essere nuovamente interrotta. Esso può essere modificato dal programma secondo le necessità. Si rimanda per un approfondimento al paragrafo sulle interruzioni.
L'ultimo gruppo comprende 3 flag che vengono trattati automaticamente dalla CPU e non sono modificabili dal programma: sono il flag di zero (Z), il flag di segno (N) e il flag di break (B).
Il flag Z viene messo a 1 tutte le volte che il risultato di una operazione è zero.
Il flag N copia lo stato del bit più significativo del risultato di un'operazione; viene quindi messo a 1 se il risultato è negativo, a 0 se il risultato è positivo.
Infine il flag B viene messo a 1 durante la sequenza di servizio di una interruzione software.
I flag vengono usati nella stesura di programmi in linguaggio macchina per realizzare i cosiddetti salti condizionati, per mezzo dei quali si fanno seguire a un programma percorsi diversi a seconda dello stato di un flag; i salti condizionati sono l'equivalente dell'istruzione BASIC:

IF...THEN GOTO nn

Il registro puntatore dell'area di stack (S) contiene l'indirizzo di un byte in pagina 1; il valore 0 corrisponde alla locazione 256, il valore 1 alla locazione 257 e così via fino a 255 che corrisponde alla locazione 511. Il byte indirizzato dal registro S rappresenta la prima locazione libera di una zona di memoria di uso particolare, che si chiama "area di stack", a disposizione sia del programma che del sistema per depositarvi temporaneamente dei dati o degli indirizzi.
L'area di stack può essere immaginata come una "pila" (questo sarebbe il termine

esatto in italiano) di byte uno sopra l'altro, nel senso che, a partire da un certo indirizzo, che può essere al massimo 511, i byte depositati in questa zona vengono accumulati uno sopra l'altro, nella direzione degli indirizzi bassi di memoria. L'indirizzo del byte più alto della pila, che risulta l'indirizzo più basso della pila, non può essere inferiore a 256.
L'area di stack e il suo funzionamento sono illustrati nella figura che segue.

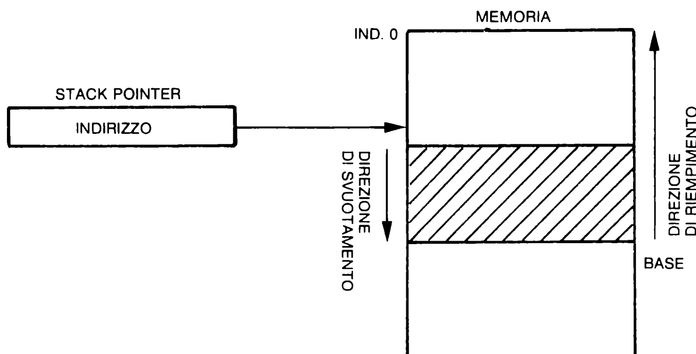


Figura 1.9 Area di stack e puntatore

Così come l'inserimento nell'area di stack non può avvenire che aggiungendo byte nella direzione degli indirizzi più bassi, il prelevamento non può avvenire che nell'ordine inverso; questa tecnica di gestione prende il nome di LIFO (Last In First Out) e significa che il primo dato ad essere prelevato non può essere che l'ultimo inserito.
Ad ogni operazione di inserimento di un byte, il registro S viene automaticamente decrementato dal sistema, mentre ad ogni operazione di prelevamento viene incrementato, in modo da puntare sempre alla prima locazione libera.
L'area di stack è a disposizione dell'utente, ed è comoda da usare per conservarvi il contenuto di registri che devono essere provvisoriamente usati per altre funzioni; inoltre viene utilizzata automaticamente dalla CPU nella gestione delle interruzioni e dei sottoprogrammi, come si vedrà in seguito.
L'Accumulatore è un registro a 8 bit di uso generale; ciò significa che non è dedicato a una funzione particolare, ma viene usato in diverse operazioni, in genere per depositarvi provvisoriamente risultati intermedi o finali. Inoltre viene sempre usato per trasferimenti di dati da una zona a un'altra di memoria, cosa che non può essere fatta direttamente: il dato da spostare deve essere prima trasferito dalla locazione originaria all'accumulatore, e da questo nella locazione finale.
I registri indice X e Y sono utilizzati per realizzare la tecnica di indirizzamento indicizzato, che si descrive dettagliatamente più avanti.

1.4 UNITA' ARITMETICA E RAPPRESENTAZIONE INTERNA DEI NUMERI

Il 6502 è un microprocessore a 8 bit; questo significa che un'operazione aritmetica o logica tratta dati di 8 bit (un byte). Le operazioni aritmetiche o logiche vengono eseguite da quella parte della CPU che prende il nome di Unità Aritmetico Logica (ALU); i dati su cui esse operano si chiamano operandi, e si possono trovare in memoria o nei registri. La tecnica con cui vengono individuati gli operandi va sotto il nome di "modo di indirizzamento" e si tratterà nel paragrafo seguente.
Quando un'operazione coinvolge due operandi, uno di essi si trova in memoria e l'altro nell'accumulatore; il risultato si genera nell'accumulatore e poi deve essere trasferito in memoria. Si è già visto che il trasferimento di un dato avviene sempre tramite l'accumulatore e non direttamente tra due posizioni di memoria.
I circuiti della ALU eseguono via hardware addizioni e sottrazioni, e un certo numero di operazioni logiche, come OR, AND e altre, a cui corrispondono altrettante istruzioni in linguaggio macchina. Non esistono invece, nel linguaggio macchina, istruzioni per eseguire moltiplicazione e divisione e altre operazioni aritmetiche, che si possono realizzare con programmi scritti dall'utente.
Nella programmazione in linguaggio macchina si possono trattare numeri con e senza segno; quelli senza segno si possono rappresentare in binario assoluto o in BCD, quelli con segno in complemento a 2. I numeri in binario assoluto sono numeri senza segno codificati in binario; in un byte si possono rappresentare numeri da 0 a 255. Quando il risultato di un'operazione supera 255, il flag del Carry viene posto a 1 per segnalare un riporto.
In BCD i numeri sono codificati cifra per cifra, usando 4 bit per ogni cifra e codificandola in binario. In questo modo ogni byte contiene 2 cifre e quindi si possono rappresentare numeri da 00 a 99. Si possono eseguire istruzioni in BCD; in questo caso il flag di Carry viene posto a 1 se il risultato supera 99.
Infine i numeri con segno si rappresentano in complemento a 2; con questa tecnica, delle 256 configurazioni possibili in un byte, la metà viene utilizzata per numeri positivi, l'altra metà per numeri negativi, e precisamente le 128 configurazioni che hanno il MSB (Most Significant Bit) a 0 rappresentano 128 numeri positivi, da 0 a 127, in binario puro; le 128 configurazioni che hanno il MSB a 1 rappresentano 128 numeri negativi, da -1 a -128. Il modo in cui vengono rappresentati i numeri negativi è il seguente: si scrive il numero positivo in 8 bit (con il MSB a 0), lo si complementa bit a bit, cioè si cambiano gli 1 in 0 e viceversa, e infine si somma 1. Ad esempio, per rappresentare -38, si scrive 38 in 8 bit in binario:

00100110

che complementato diventa
11011001

infine sommando 1 si ha
11011001 + 1 = 11011010

in conclusione
$-38_{10} = 11011010_2$

Per passare dalla rappresentazione di un numero negativo in complemento a 2 al suo valore decimale basta sviluppare, come al solito, il numero binario in somme di prodotti dei singoli bit per la potenza del 2 corrispondente alla posizione, sottraendo però il prodotto relativo all'ultimo bit. Per l'esempio visto si ha:

$$11011010_2 = 2^1 + 2^3 + 2^4 + 2^6 - 2^7 =$$
$$2 + 8 + 16 + 64 - 128 = -38_{10}$$

Quando si trattano numeri con segno, non è più il flag di Carry (C), ma il flag di Overflow (V), a segnalare il superamento della capacità di un byte nel risultato di un'operazione. E' importante capire, a questo proposito, che il calcolatore "ignora" se i numeri al suo interno sono rappresentati con o senza segno, cioè in binario puro o in complemento a 2, e i flag C e V vengono modificati in modo automatico in conseguenza di certi eventi, indipendentemente dalla rappresentazione adottata dal programma; il flag C quando c'è un riporto dal bit 7 del risultato, il flag V quando c'è un riporto dal bit 6 (i bit si contano da destra verso sinistra come bit 0, 1,...,7, quindi il bit 0 è il primo e il bit 7 è l'ottavo). La scelta fra una rappresentazione con o senza segno riguarda esclusivamente il programmatore e l'elaborazione che egli vuol fare; sarà sua cura controllare il flag giusto per non fare errori. Per la ALU le regole per eseguire le operazioni sono sempre le stesse sia in presenza che in assenza di segno nei numeri.
Per i numeri in BCD invece si usano regole diverse, ma il flag da controllare è sempre il C.
In tutte e 3 le rappresentazioni viste, la grandezza dei numeri trattabili è molto limitata; per poter eseguire operazioni su numeri più grandi si devono scrivere delle routine che, sfruttando le istruzioni in linguaggio macchina, svolgano i calcoli necessari. Routine di questo tipo sono per esempio contenute nell'interprete BASIC, dato che nel linguaggio si possono trattare facilmente anche numeri grandi.

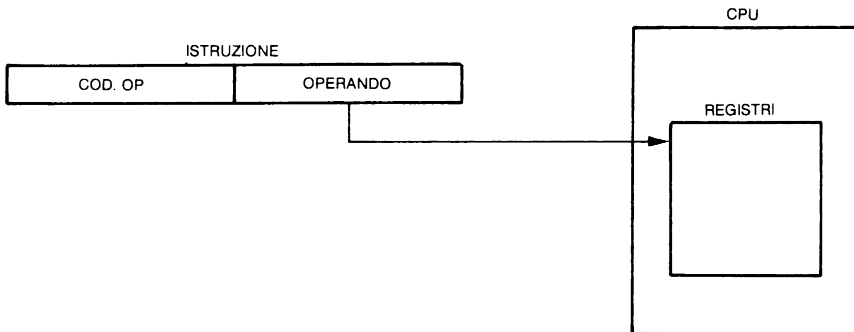
1.5 MODI DI INDIRIZZAMENTO

Le istruzioni macchina del 6502 sono costituite da 2 campi: codice operativo e operando. Il codice operativo occupa un byte, e, una volta decodificato, fornisce alla CPU le informazioni necessarie per l'esecuzione della istruzione. Il campo operando contiene le informazioni necessarie a identificare il dato su cui l'istruzione deve operare; i diversi modi con cui viene definito il campo operando prendono il nome di "modi di indirizzamento". Essi sono:

Implicito	Assoluto indicizzato tramite reg. X
Tramite accumulatore	Assoluto indicizzato tramite reg. Y
Immediato	Indicizzato in pag. 0 tramite reg. X
Assoluto	Indicizzato in pag. 0 tramite reg. Y
In pagina 0	Indiretto indicizzato
Relativo	Indicizzato indiretto
Indiretto	

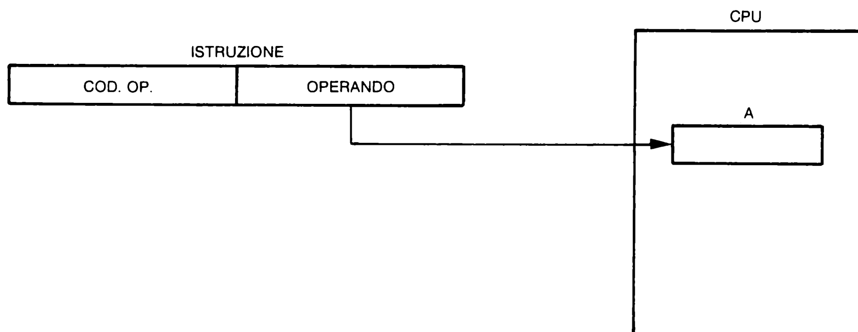
INDIRIZZAMENTO IMPLICITO

Il codice operativo contiene implicitamente al suo interno le informazioni sulla posizione dell'operando, che in questo caso è il contenuto di un registro della CPU. Le istruzioni che usano questo tipo di indirizzamento sono costituite da un solo byte, che riassume al suo interno le due funzioni di codice operativo e operando.



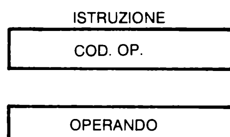
TRAMITE ACCUMULATORE

E' un caso particolare dell'indirizzamento implicito; l'operando si trova nell'accumulatore, e l'istruzione è costituita da un unico byte.



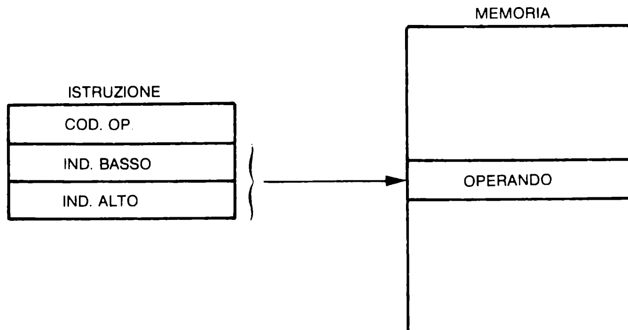
IMMEDIATO

L'operando è una costante, il cui valore viene scritto nel byte successivo a quello del codice operativo: l'istruzione è a 2 byte. L'inconveniente di questo modo è che per modificare il valore del dato si deve modificare la parte istruzioni del programma.



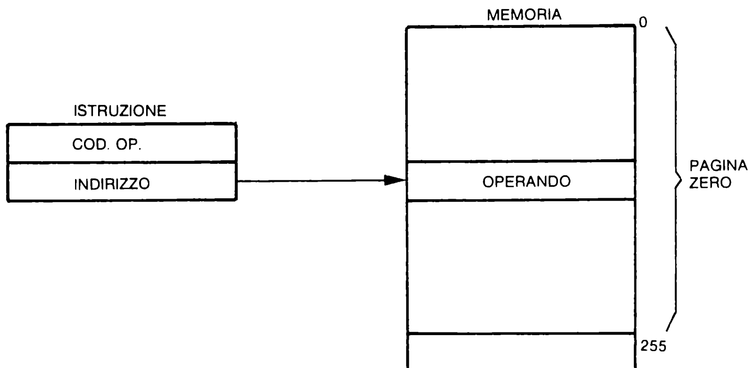
ASSOLUTO

L'operando è una variabile, che si trova in una certa locazione di memoria il cui indirizzo viene fornito nel campo operando, nei 2 byte successivi al codice operativo: l'istruzione è a 3 byte. Per modificare il valore del dato, basta modificare il contenuto della locazione il cui indirizzo compare nella istruzione, senza toccare la parte istruzioni del programma; da qui il nome di variabile.



IN PAGINA ZERO

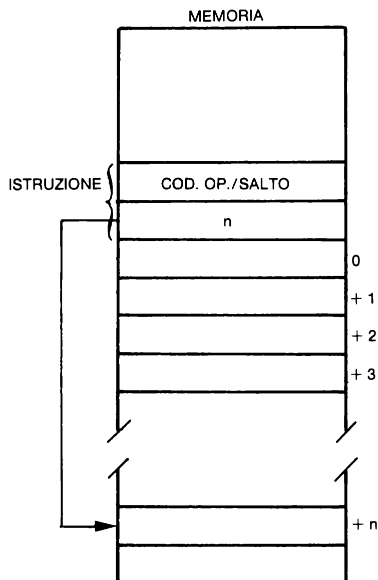
E' un caso particolare di indirizzamento assoluto. L'operando si trova in una locazione della pagina zero; l'indirizzo che compare nell'istruzione occupa un solo byte e può variare da 0 a 255, indirizzi dei byte della pagina zero.

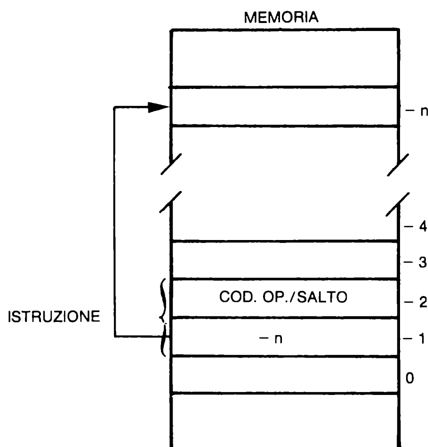


Con questo modo di indirizzamento, e con tutti quelli che usano la pagina zero, occorre tener presente che molte locazioni di questa pagina sono utilizzate dal sistema e quindi non possono essere usate dal programmatore per i suoi programmi in linguaggio macchina.

RELATIVO

E' utilizzato esclusivamente dalle istruzioni di salto, che provocano l'interruzione dell'esecuzione sequenziale del programma e il trasferimento del controllo ad una locazione il cui indirizzo viene indicato nell'istruzione stessa. L'operando occupa



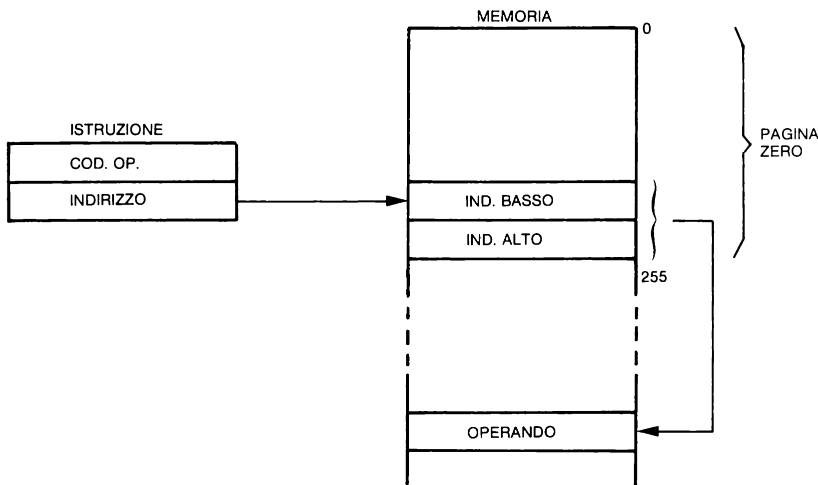


un byte, e contiene un numero con segno che indica la distanza in byte dell'istruzione a cui si deve saltare, da quella immediatamente successiva all'istruzione di salto. Questo numero prende il nome di "spiazzamento" e può variare da -128 a +127; se positivo il salto è in avanti, se negativo a ritroso. In sostanza lo spiazzamento viene

sommato dalla CPU al Program Counter prima di procedere nell'esecuzione del programma. Si tratta di istruzioni a 2 byte.

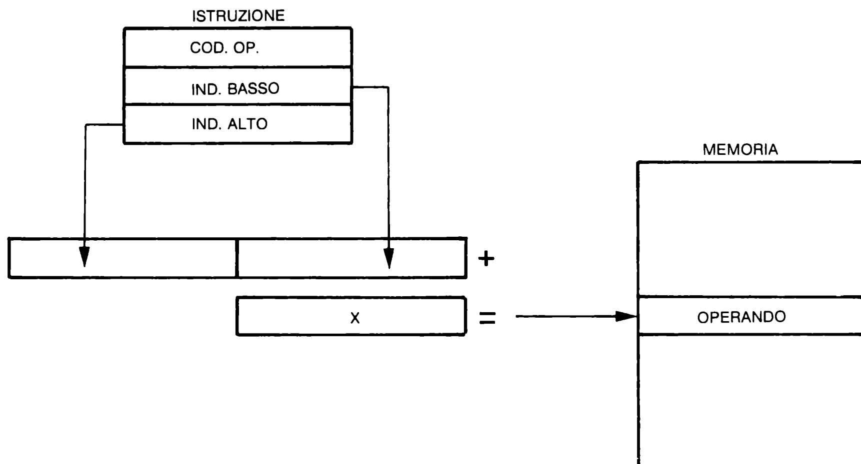
INDIRETTO

L'operando, di 1 byte, contiene l'indirizzo (in pagina zero) della prima di 2 locazioni contigue, che, a loro volta, contengono l'indirizzo del dato. E' utilizzato dalle istruzioni di salto con codice simbolico JMP, che occupano 2 byte.



ASSOLUTO INDICIZZATO TRAMITE REGISTRO X

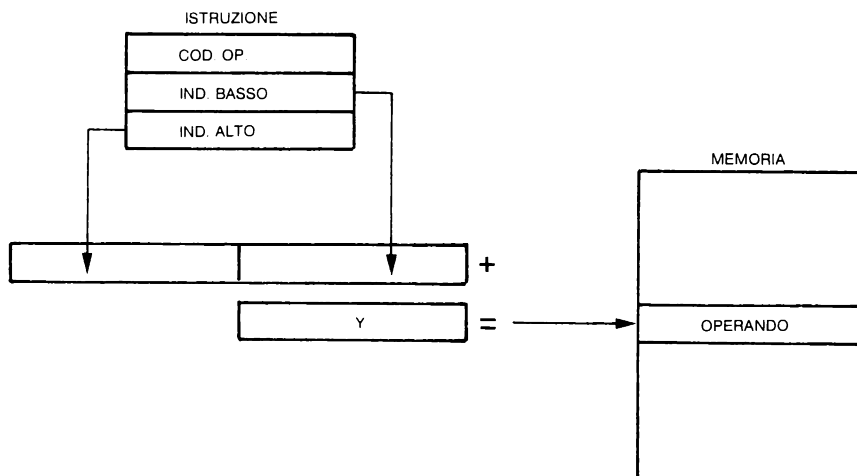
L'operando, di 2 byte, contiene un valore che viene sommato al contenuto del registro X per ottenere l'indirizzo del dato su cui operare. Questo modo si presta



all'elaborazione di blocchi di dati scritti in un certo numero di byte contigui; il registro X fa da puntatore alla posizione del byte all'interno del blocco, e viene incrementato ciclicamente per consentire l'accesso sequenziale a tutti i byte. Le istruzioni sono a 3 byte.

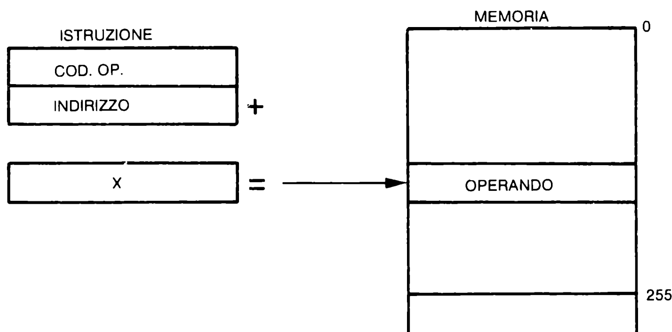
ASSOLUTO INDICIZZATO TRAMITE REGISTRO Y

E' identico al precedente, ma utilizza il registro Y.



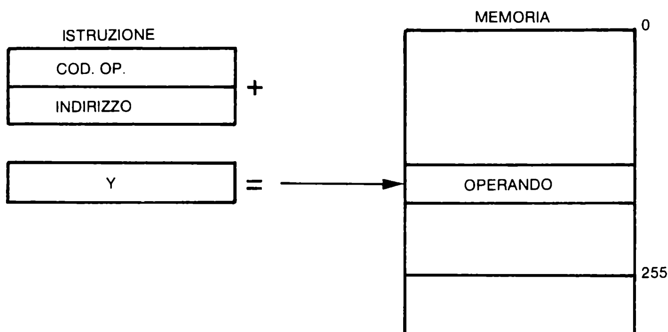
INDICIZZATO IN PAGINA ZERO TRAMITE REGISTRO X

E' analogo al precedente, con la differenza che l'operando occupa un solo byte, e contiene un indirizzo di pagina zero. Le istruzioni sono a 2 byte.



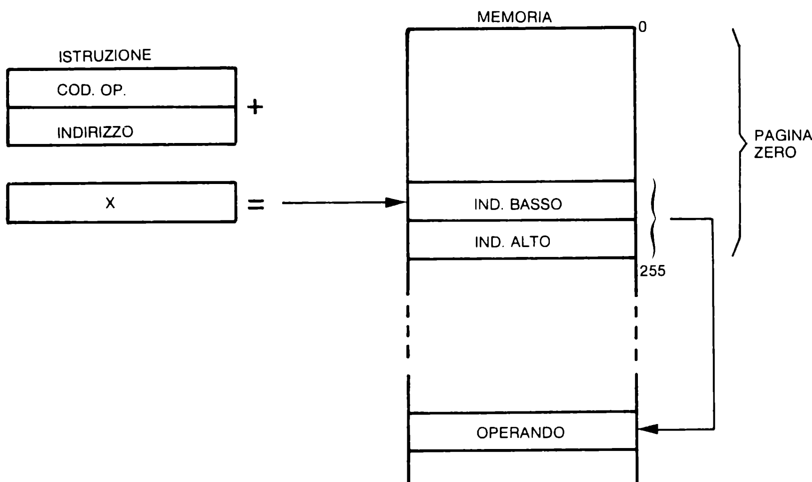
INDICIZZATO IN PAGINA ZERO TRAMITE REGISTRO Y

E' identico al precedente, ma opera sul registro Y.



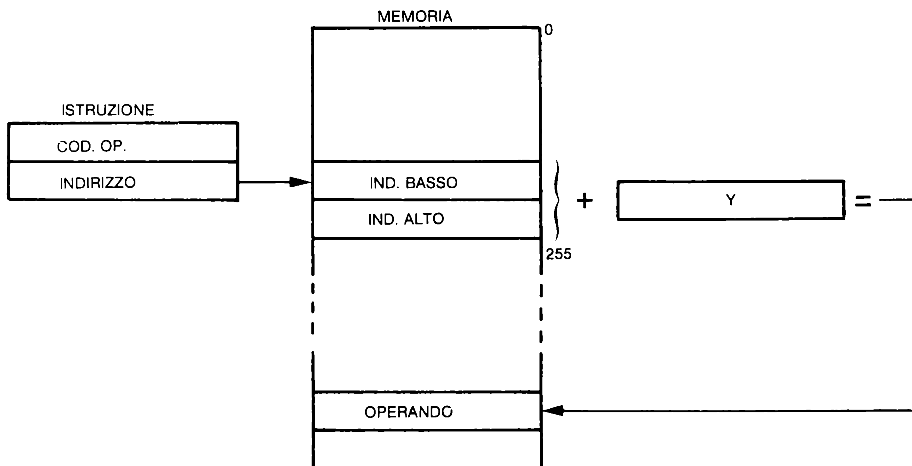
INDICIZZATO INDIRETTO

L'operando, di 1 byte, contiene un indirizzo di pagina zero; questo viene sommato al contenuto del registro X, per ottenere l'indirizzo della prima di due locazioni di memoria contigue in cui è scritto l'indirizzo effettivo dell'operando. Sono istruzioni a 2 byte. Questo modo si presta a gestire in modo semplice tabelle di indirizzi, da "spazzolare" sequenzialmente, come può capitare nell'esecuzione di programmi di "polling", cioè di interrogazione ciclica di un certo numero di dispositivi esterni, i cui indirizzi sono appunto elencati in una lista.



INDIRETTO INDICIZZATO

L' operando, di 1 byte, contiene un indirizzo di pagina zero; in quella locazione e nella successiva è scritto un numero che, sommato al contenuto del registro Y, fornisce l'indirizzo effettivo del dato. Questo modo può servire per accedere ai dati di una tabella collocata in un qualsiasi punto della memoria, il cui indirizzo iniziale è scritto in pagina zero.



1.6 SOTTOPROGRAMMI

Un sottoprogramma (subroutine) è una sequenza di istruzioni che vengono scritte e registrate in una zona di memoria una volta per tutte, e possono essere eseguite più

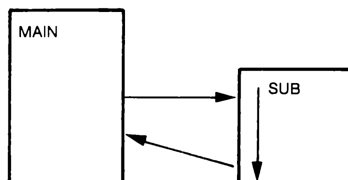


Figura 1.10 Chiamata e ritorno da sottoprogramma

volte, durante l'esecuzione di un programma, trasferendo, con una opportuna istruzione, il controllo della CPU al primo indirizzo del sottoprogramma stesso.

L'istruzione con cui viene mandato in esecuzione un sottoprogramma prende il nome di "chiamata al sottoprogramma".

Ogni sottoprogramma deve terminare con un'istruzione detta di "return" (ritorno dal sottoprogramma), che rimanda il controllo della CPU al punto del programma in cui il sottoprogramma era stato chiamato, e precisamente all'istruzione successiva a quella di chiamata.

Si è già accennato al fatto che l'area di stack e il registro puntatore alla stessa vengono utilizzati automaticamente dalla CPU nelle operazioni di chiamata e ritorno da sottoprogramma; si ritorna qui sull'argomento. Un sottoprogramma è identificato dal suo indirizzo di inizio; l'istruzione di chiamata è un salto "speciale" a questo indirizzo. Speciale perchè, oltre a caricare nel Program Counter l'indirizzo del sottoprogramma, e quindi provocare il salto, esso svolge l'operazione di memorizzare l'indirizzo successivo a quello della istruzione di salto. La successione delle operazioni è la seguente: prima del salto, il Program Counter contiene l'indirizzo dell'istruzione successiva (come sempre), e tale indirizzo va nell'area stack con conseguente aggiornamento del suo puntatore. A questo punto, nel PC viene caricato l'indirizzo di salto; il sottoprogramma va in esecuzione e, quando la CPU incontra l'istruzione di "ritorno da sottoprogramma", va a prelevare dall'area stack l'indirizzo che vi aveva memorizzato e lo rimette nel Program Counter, facendo avvenire un salto di ritorno. Il meccanismo descritto è illustrato dalla figura 1.10.

Tutte queste operazioni sono assolutamente trasparenti all'utente, nel senso che vengono svolte automaticamente dalla CPU, che garantisce in questo modo il corretto funzionamento del programma e dei sottoprogrammi.

Occorre prestare attenzione a un particolare; quando il sottoprogramma preleva l'indirizzo di ritorno dall'area stack, va a prendere i due byte disponibili al momento, in base al valore del puntatore. Se all'interno del sottoprogramma non è stato fatto un uso bilanciato dell'area stack, nel senso che tutto quello che è stato messo è stato anche prelevato, alla fine i due byte affioranti non contengono l'indirizzo di rientro e ciò provoca notevoli errori.

Quando all'interno di un sottoprogramma si ha la chiamata a un altro sottoprogramma, nell'area stack si trovano, uno sopra l'altro due indirizzi di ritorno; il sottoprogramma chiamato per ultimo deve finire prima del precedente. Nella figura 1.11 si schematizza questa situazione.

Questa struttura di programma si chiama "a subroutine nidificate"; il numero di subroutine che possono essere nidificate è limitato dalle dimensioni dell'area di stack. Infatti, tenendo conto che ogni chiamata a sottoprogramma occupa due byte in tale area, per il salvataggio del contenuto del Program Counter, è evidente che non si possono avere più di 128 subroutine nidificate; questo limite è largamente accettabile nei programmi che vengono usualmente implementati sul calcolatore.

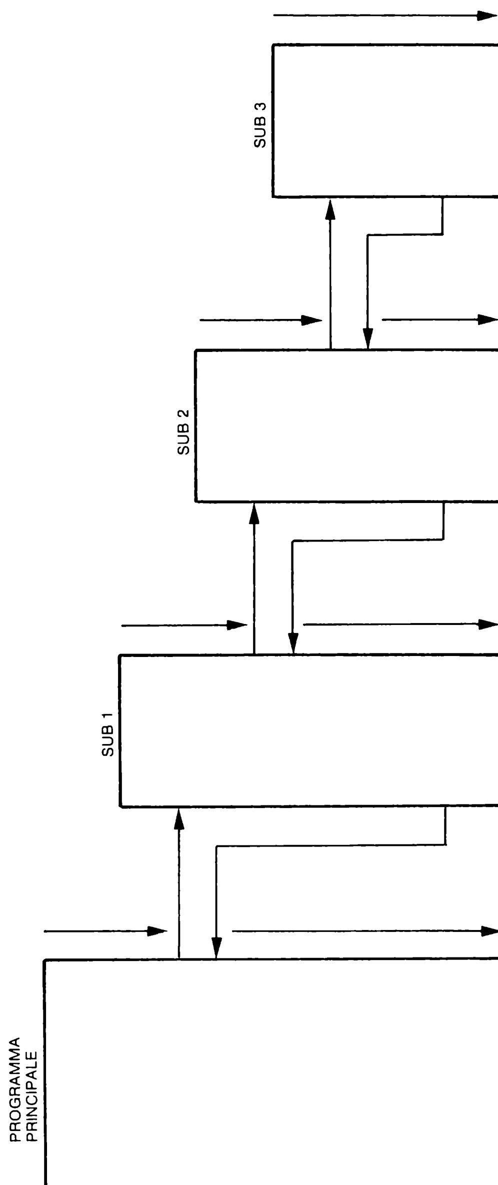


Figura 1.11 Sottoprogrammi concatenati o nidificati

1.7 INTERRUPT

La CPU usa l'area di stack anche per un'altra operazione: la gestione degli interrupt.

La comunicazione tra la CPU e le periferiche può avvenire con due tecniche diverse. La prima prende il nome di "polling" (interrogazione ciclica): la CPU interroga ciclicamente, secondo un ordine prestabilito, i dispositivi esterni, usando un programma che legge lo stato degli stessi. Quando uno dei dispositivi è pronto a trasmettere o a ricevere un dato, la CPU manda in esecuzione il programma neccessario all'operazione richiesta. La priorità tra i diversi dispositivi è determinata dall'ordine con cui vengono interrogati.

Questa tecnica presenta un solo vantaggio, il fatto di essere realizzata completamente via software, e quindi di non richiedere circuiteria addizionale per le comunicazioni. Presenta però diversi svantaggi, che in alcuni casi la rendono addirittura impraticabile: la CPU è praticamente occupata per la maggior parte del tempo a interrogare i dispositivi, mentre solo una minima parte di questo tempo è impiegato per la comunicazione vera e propria. Inoltre il tempo di risposta della CPU alle richieste dei dispositivi può essere in certi casi troppo lungo; se ad esempio i dispositivi sono parecchi, e uno di essi richiede un'operazione di I/O immediatamente dopo essere stato interrogato, deve aspettare parecchio tempo prima di essere servito; in certi casi questo può comportare la perdita di dati.

La seconda tecnica, che elimina questi svantaggi, è quella degli interrupt. I dispositivi stessi segnalano alla CPU la richiesta di un'operazione di I/O, inviando un segnale, che si chiama appunto interrupt (o interruzione), per richiedere alla CPU di interrompere il programma che sta eseguendo, e eseguire l'operazione di I/O. Il servizio di un interrupt avviene con un salto ad una routine, che prende il nome di routine di servizio dell'interrupt, che provvede ad eseguire l'operazione richiesta.

Questa tecnica presenta molte analogie con l'esecuzione delle subroutine, con la differenza che l'esecuzione della routine di servizio avviene in momenti non prevedibili dal programma, che dipendono dalle esigenze dei dispositivi esterni.

E' chiaro che la tecnica degli interrupt elimina gli svantaggi cui si è prima accennato. Il tempo di risposta è limitato solo dalla velocità della CPU a trasferire il controllo da una zona a un'altra della memoria; inoltre la CPU può dedicarsi ad altri programmi, utilizzando in modo più efficiente il suo tempo. Per chiudere un paragone significativo: la differenza tra polling e interrupt è la stessa che ci sarebbe tra aprire la porta di casa ad intervalli di tempo prefissati, per vedere se c'è qualcuno, e il rispondere ad un trillo del campanello. La perdita di tempo del primo caso è evidente, come è evidente la scomodità del servizio per chi, desiderando comunicare, deve attendere il prosssimo controllo per poterlo fare.E per restare in questo esempio, così come nel secondo caso è però necessario un dispositivo (il campanello) per segnalare la richiesta di comunicazione, per la gestione delle interruzioni è necessario un hardware più complicato. Prima di tutto occorrono una o più linee della CPU dedicate al ricevimento dei segnali di interrupt; in più occorre che il dispositivo interrompente si faccia riconoscere dalla CPU, e anche

questo rende più complessi i collegamenti; infine, anche se non sempre, è necessaria della circuiteria che consenta il servizio dei diversi dispositivi in base alla priorità prefissata.
Si fa notare che anche nel caso delle interruzioni la CPU esegue una specie di polling; infatti legge periodicamente, in genere alla fine di ogni istruzione, la linea di interrupt per sapere se è stata fatta una richiesta di I/O. Questa operazione, però, a differenza del polling, viene eseguita dallo hardware stesso della CPU, e porta via un tempo trascurabile all'esecuzione del programma in corso.
Spesso la tecnica di gestione delle comunicazioni con le periferiche è mista, nel senso che utilizza sia le interruzioni che il polling; ad esempio, nel caso che diversi dispositivi siano collegati ad un'unica linea di interrupt, non appena uno di essi richiede un servizio la CPU si interrompe, e la routine di servizio che va in esecuzione esegue un polling sui dispositivi per identificare quello chiamante.

Nel 6502 ci sono 2 linee di interrupt, la linea IRQ (Interrupt Request), e la linea NMI (Not-maskable interrupt). Entrambe queste linee sono "attive basse"; ciò significa che la segnalazione di un interrupt si effettua mettendole a zero, per almeno 20 microsecondi.
Alla linea NMI vanno collegati quei dispositivi le cui richieste devono essere servite immediatamente e con la massima priorità; un segnale di interrupt su questa linea, infatti, non può essere ignorato dalla CPU che viene interrotta in ogni caso.
Alla linea IRQ, invece, vanno collegati quei dispositivi le cui richieste possono o meno essere servite, a seconda della situazione.
Un segnale di interruzione su una di queste linee viene sentito dalla CPU alla fine dell'istruzione in corso, questo per evitare che l'esecuzione dell'istruzione venga sospesa, il che renderebbe problematica la ripresa del programma. La CPU sospende l'esecuzione del programma in corso, salva sia il Program Counter che il registro di stato nell'area stack, aggiornando automaticamente il relativo puntatore, e trasferisce il controllo all'indirizzo che si trova in una certa locazione di memoria. A questo indirizzo si trova la routine di servizio dell'interrupt; analogamente ai sottoprogrammi, essa termina con un'istruzione di ritorno, che ripristina nel Program Counter e nel registro di stato i valori che erano stati conservati nell'area di stack prima della sua esecuzione. In questo modo il programma interrotto riprende la sua esecuzione in modo corretto.
Quando arriva un interrupt sulla linea NMI, la CPU salta ad una routine di servizio il cui indirizzo si trova in ROM nelle locazioni 65530 e 65531 (FFFAH-FFFBH), mentre quando arriva sulla linea IRQ l'indirizzo viene prelevato dalle locazioni 65534 e 65535 (FFFEH-FFFFH). Nel VIC le routine di servizio che sono in ROM svolgono alcune operazioni necessarie, come il salvataggio dello stato del calcolatore, prima di eseguire un salto indiretto a un indirizzo della RAM: 788 (0314H) per IRQ e 792 (0318H) per NMI. In queste locazioni è scritto l'indirizzo effettivo di inizio delle routine di servizio; così il programmatore può modificare la gestione degli interrupt, scrivendo delle routine personali e modificando gli indirizzi contenuti in RAM (vettori di interrupt).

La linea NMI nel VIC è collegata alla porta VIA n.1, e la IRQ alla PORTA VIA n.2, e le routine di servizio corrispondenti gestiscono le comunicazioni con queste porte di I/O, come si vedrà in seguito.
Per precisare ulteriormente la differenza tra interrupt mascherabile e non mascherabile si consideri che nel registro di stato si ha un bit, detto flag I di interrupt; questo flag può essere settato e resettato da programma con opportune istruzioni, e condiziona l'accettazione o meno di un interrupt sulla linea IRQ da parte della CPU. Quando il flag è a 1 gli interrupt sono mascherati, cioè l'eventuale attivazione della linea IRQ viene ignorata dalla CPU; quando è a 0, vengono invece sentiti e serviti.
Il programmatore può quindi decidere se e quando il programma deve essere interrotto; inoltre può evitare o meno che una routine di servizio di un interrupt sia interrotta a sua volta da un altro dispositivo, a seconda delle esigenze dell'applicazione. In pratica si possono stabilire dei livelli di priorità via software. Va tenuto presente che la CPU stessa pone a 1 il flag I non appena riceve un'interruzione, e lo pone a zero quando finisce l'esecuzione della routine di servizio; quindi, salvo diverse disposizioni del programmatore, una routine di servizio non può essere interrotta.
Questo discorso non vale per i segnali di interrupt che arrivano sulla linea NMI; il flag I non ha alcuna influenza su di essi, e quindi vengono serviti in ogni caso. Per questo vengono chiamati interrupt non mascherabili. Di fatto gli interrupt sulla linea NMI hanno priorità più alta di quelli sulla linea IRQ.
Il 6502 ha inoltre la caratteristica di poter generare un interrupt via software, tramite l'istruzione BRK. Essa provoca un salto della CPU all'indirizzo che si trova nelle locazioni assegnate alla linea IRQ, e setta il flag B del registro di stato; in questo modo la stessa routine di servizio è in grado di riconoscere se l'interrupt è stato generato da software o da hardware, e seguire due strade diverse nei due casi. L'uso di questa istruzione è però consigliato solo a programmatori molto esperti.

CAPITOLO 2

GESTIONE DEL VIDEO

2.1 MEMORIA PER LA GESTIONE DEL VIDEO

La gestione del video collegato al VIC è affidata ad un chip di interfaccia dalle molteplici e complesse funzioni, il 6561 della Motorola.
Grazie a questo chip è possibile realizzare grafica ad alta risoluzione, colorare in modo "variopinto" l'immagine sul video, disegnare caratteri particolari definiti dall'utente. Inoltre si possono generare suoni, effettuare una conversione analogico/digitale per i joystick, e usare una "light pen".

Il chip 6561, utilizza, per la gestione del video, tre aree di memoria:

.a) la memoria di schermo (MAPPA VIDEO),
.b) la memoria colore (MAPPA COLORI),
.c) la memoria caratteri (MAPPA CARATTERI).

Nel seguito si usano le espressioni tra parentesi per individuare le parti a), b) e c).

La MAPPA VIDEO è una zona di 506 byte in RAM; ogni byte corrisponde ad una delle posizioni dell'immagine sul video; il primo byte corrisponde alla prima posizione in alto a sinistra, il secondo alla posizione successiva sulla stessa linea, e così via.
Si ricorda che le dimensioni del quadro video sono di 23 righe di 22 caratteri, per un totale appunto di 506 caratteri.
Ogni byte contiene il codice del carattere (non in ASCII, ma in DISPLAY CODE, si veda Appendice B del I Volume sul VIC di R. B.) che deve comparire nella posizione corrispondente; esso fa da puntatore ad una zona della mappa caratteri, in cui è definito il carattere stesso.
La MAPPA VIDEO inizia alla locazione 7680 (1E00H) nella configurazione standard del VIC e in quella con 3K di espansione, mentre viene spostata alla locazione 4096 (1000H) nella versione con espansioni di memoria superiori a 3K.
La posizione della mappa video è controllata da alcuni bit dei registri 2 e 6 del 6561, come si vedrà più avanti; inoltre al momento dell'accensione del calcolatore il byte

648 contiene 30 o 16 che, moltiplicati rispettivamente per 256, danno l'indirizzo di inizio di questa mappa (30*256=7680, 16*256=4096).
La MAPPA COLORI è una zona di 506 byte su RAM, ognuno dei quali corrisponde, anche in questo caso, ad una posizione del video, con lo stesso criterio visto per la mappa video. I 3 bit meno significativi di ogni byte (bit 0-2) contengono il codice di uno tra 8 colori disponibili; il bit 3 seleziona il modo "alta risoluzione" quando è a 0 (e questa è la norma), e il modo "multicolor" quando è a 1. Nel primo caso il colore indicato dai bit 0-2 viene assegnato al carattere o al suo sfondo, in dipendenza dallo stato del bit 3 del registro 16, di indirizzo 36879 (900FH), del 6561. Si vedrà più avanti come viene utilizzato il contenuto dei byte della mappa colori nel modo "multicolor". La mappa colori inizia alla locazione 38400 (9600H nella configurazione standard del VIC e in quella con 3K di espansione, mentre viene spostata alla locazione 37888 (9400H) nelle altre configurazioni.
La MAPPA CARATTERI è la zona di memoria in cui sono descritti, punto per punto, i diversi caratteri da visualizzare. Il set standard di caratteri del VIC è definito in ROM a partire dalla locazione 32768 (8000H), ma l'utente può definire in RAM una nuova mappa caratteri in base alle sue esigenze.
Nella mappa caratteri standard, disponibile all'accensione del calcolatore, ogni carattere viene descritto in 8 byte consecutivi, come nella figura seguente:

	bit	7	6	5	4	3	2	1	0
byte	32776	0	0	0	1	1	0	0	0
	32777	0	0	1	0	0	1	0	0
	32778	0	1	0	0	0	0	1	0
	32779	0	1	0	0	0	0	1	0
	32780	0	1	1	1	1	1	1	0
	32781	0	1	0	0	0	0	1	0
	32782	0	1	0	0	0	0	1	0
	32783	0	0	0	0	0	0	0	0

Figura 2.1 Rappresentazione in memoria della lettera A

Ogni bit a 0 corrisponde a uno spazio, ogni bit a 1 ad un punto sul video. Ogni carattere viene descritto per punti in una griglia di 8x8 bit.

Nella figura 2.2 è schematizzato il meccanismo di visualizzazione di un carattere, che si può così riassumere: ad ogni posizione dello schermo corrisponde un byte nella mappa video e un byte nella mappa colori; quest'ultimo definisce il colore con cui il carattere deve comparire in quella posizione, mentre il byte della mappa video contiene un puntatore alla zona della mappa caratteri, dove è descritto il carattere. I caratteri nella mappa caratteri sono messi in corrispondenza con un codice crescente che parte da zero (display code) e che compare come puntatore

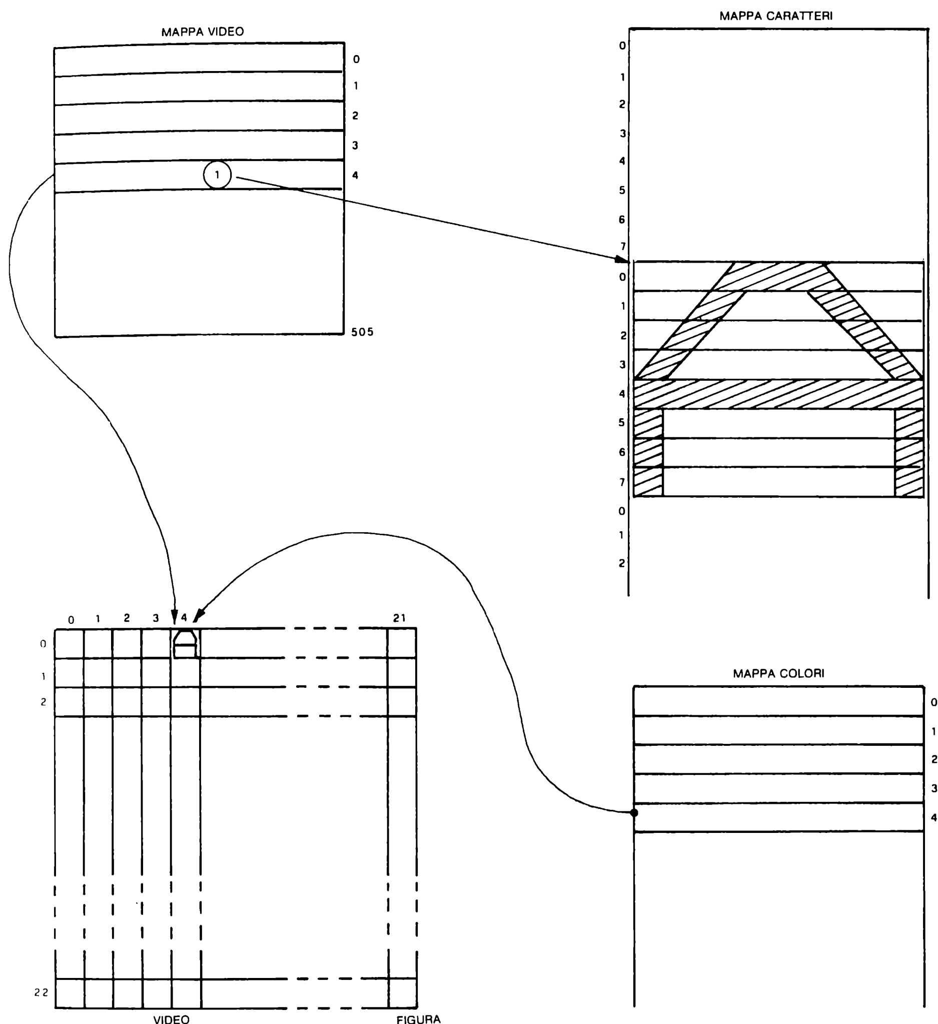


Figura 2.2 Visualizzazione di un carattere sul video

nella mappa video. L'indirizzo effettivo del primo byte di descrizione del carattere è ricavato con un semplice calcolo, moltiplicando per 8 il codice e sommandogli l'indirizzo di inizio della mappa caratteri.
Si ricorda che per far comparire un carattere sul video occorre scrivere sia il codice del carattere nella mappa video, che il codice del colore nella mappa colori. Se si dimentica il codice colore non si vede il carattere; infatti il sistema, al momento dell'accensione, assegna il colore bianco sia ai caratteri che allo sfondo.

La mappa caratteri standard occupa 4K byte, in cui sono descritti 512 caratteri standard. Essi sono nell'ordine:

-128 caratteri maiuscoli e grafici,
-128 caratteri maiuscoli e grafici in campo inverso,
-128 caratteri maiuscoli e minuscoli,
-128 caratteri maiuscoli e minuscoli in campo inverso.

I primi 256 caratteri, per un totale di 2K, costituiscono quello che si chiama il set Grafico del VIC; gli altri 256 sono il set Maiuscole/Minuscole. Al momento dell'accensione viene selezionato il set Grafico, mentre premendo contemporaneamente i tasti SHIFT e COMMODORE si passa all'altro. Altri modi per passare da un set di caratteri all'altro sono i seguenti:

POKE 36869,240 set Grafico
POKE 36869,242 set Maiusc./Minusc.

se si lavora con configurazione standard o con espansione fino a 3K;

POKE 36869,192 set Grafico
POKE 36869,194 set Maiusc./Minusc.

se si lavora con espansioni superiori a 3K;

PRINT CHR$(14) set Grafico
PRINT CHR$(142) set Maiusc./Minusc.

con qualunque configurazione.

L'indirizzo di partenza della mappa caratteri può essere modificato dall'utente, tramite il registro 6 (di indirizzo 36869); questo consente di costruire in RAM una mappa caratteri personalizzata e di istruire il calcolatore ad usarla in alternativa alla mappa standard.
Per disabilitare il funzionamento dei tasti SHIFT e COMMODORE si può scrivere PRINT CHR$(8). Scrivendo PRINT CHR$(9) si riabilitano gli stessi stati (lo stesso effetto prodotto da RUN (STOP + RESTORE) CHR$(8) pone nel byte 657 il valore 0, mentre CHR$(9) pone nel byte 657 il valore 128.

2.2 REGISTRI INTERNI DEL 6561

Il chip 6561 ha al suo interno 16 registri a 8 bit, accessibili e programmabili dall'utente sia in BASIC che in linguaggio macchina; ogni registro è indirizzabile tramite un indirizzo di memoria, come tutti i dispositivi di I/O del VIC.

REGISTRO 1-Indirizzo 36864 (9000H)

Contenuto abituale = 12

Bit 0-6 : distanza della prima colonna di caratteri dal bordo sinistro dello schermo. Ogni incremento di 2 provoca lo spostamento del quadro di uno spazio verso destra; il valore 0 fa comparire sull'estremo sinistro la quinta colonna dell'immagine. Segue un programma esemplificativo. Alla linea 3 sono stampate 22 cifre per poter seguire meglio gli spostamenti. Il ciclo di attesa della linea 10 consente agevolmente di seguire l'andamento del programma.

```
1 REM X1
2 REM SPOSTAMENTI ORIZZONTALI QUADRO VIDEO
3 PRINT"℧01234567890123456789O1"
5 FORY=0TO64STEP2:POKE36864,Y:PRINTY:REM SPOSTA IL QUADRO
10 FORX=0TO1000:NEXTX:REM CICLO ATTESA
15 NEXTY
20 POKE36864,12:REM RIPRISTINA VALORE USUALE
```

Bit 7 : se posto a 1 attiva l'interlacciamento del segnale video (cioè l'invio delle linee pari e dispari separatamente).

REGISTRO 2-Indirizzo 36865 (9001H)

Contenuto abituale = 38

Bit 0-7 : distanza della prima riga di caratteri dal bordo superiore dello schermo; ogni incremento di 4 provoca uno spostamento di una riga verso il basso. Il valore della distanza può variare teoricamente da 0 a 255; valori maggiori o uguali a 152 provocano la fuoruscita del quadro verso il basso e possono essere utilizzati per "pulire" il video. Per osservare lo spostamento del quadro video si può provare il programma che segue.

```
1 REM X2
2 REM SPOSTAMENTI VERTICALI QUADRO VIDEO
5 FORY=0TO160STEP4:POKE36865,Y:REM SPOSTA IL QUADRO
10 FORX=0TO100:NEXTX:REM CICLO ATTESA
15 NEXTY
20 POKE36865,38:REM RIPRISTINA VALORE USUALE
```

REGISTRO 3-Indirizzo 36866 (9002H)

Contenuto abituale = 150

Bit 0-6 : numero colonne del quadro video, di norma 22. Il numero delle colonne può variare da un minimo di 1 (R3= 129 o 1) a un massimo di 27 (R3= 155 o 27). Dato che la mappa video è di 506 byte (23x22), per poter visualizzare un numero di colonne superiore a 22, occorre ridurre adeguatamente il numero di righe. In ogni caso un numero di colonne superiore a 22 non è più gestibile dall'editor dello schermo e quindi si perdono i normali allineamenti. Per osservare il fenomeno, si può provare il programma che segue.

```
1 REM X3
2 REM MODIFICA NUMERO COLONNE
3 Z=PEEK(36866)AND128:REM PREPARA Z IN BASE ALL'AMPIEZZA DELLA MEMORIA
5 FORY=Z+1TOZ+27:POKE36866,Y:PRINT"♥";Y:REM CAMBIA NUMERO COLONNE
10 FORX=0TO1000:NEXTX:REM CICLO ATTESA
15 NEXTY
20 POKE36866,Z+22:PRINT"♥":REM RIPRISTINA VALORE USUALE
```

Bit 7 : è utilizzato per comporre l'indirizzo di inizio della mappa video, insieme ai bit 4-7 del registro 6. Di norma questo bit è a 1, diventa 0 con espansioni superiori a 3K.

REGISTRO 4-Indirizzo 36867 (9003H)

Contenuto abituale = 174

Bit 0 : seleziona il formato dei caratteri.
0: caratteri 8x8 (8 byte per carattere)
1: caratteri 16x8 (16 byte per carattere, 16 righe e 8 colonne).
Il formato 16x8 è necessario nelle applicazioni di grafica ad alta risoluzione, per poter spaziare su tutto il video.

Bit 1-6 : numero di righe del quadro, di norma 23. Per modificare il numero di righe, si deve moltiplicare per 2 il numero desiderato e poi inserirlo al posto giusto, nel registro (AND 129), per non perdere il bit 0 e il bit 7. Il numero di righe può variare da 1 a 32, ma, anche qui, un numero di righe maggiore di 23 si può ottenere solo riducendo adeguatamente il numero delle colonne. Un numero di righe superiore a 23 non è più ben gestito dall'editor. Per confermare quanto detto si può provare il programma che segue.

```
1 REM X4
2 REM MODIFICA NUMERO RIGHE
5 Z=PEEK(36867)AND129:REM CONSERVA IL BIT 7 E IL BIT 0 IN Z
10 FORY=ZTOZ+52STEP2:POKE36867,Y:PRINT"J";Y:REM CAMBIA NUMERO RIGHE
15 FORX=0TO1000:NEXTX:REM CICLO ATTESA
20 NEXTY
25 POKE36867,Z+46:PRINT"J":REM RIPRISTINA VALORE USUALE
```

A conclusione delle considerazioni sul quadro video, si può provare il programma che segue, nel quale vengono modificati contemporaneamente il numero delle righe, il numero delle colonne e l'origine del quadro.

```
1 REM X5
2 REM MODIFICA QUADRO VIDEO
3 M1=PEEK(36866):M2=PEEK(36867):M3=PEEK(36864):REM MEM. CONT. 3 REGISTRI
4 PRINT"J0123456789012345678901":REM NUMERAZIONE CONTROLLO VIDEO
5 FORX=1TO3
7 POKE36864,M3-X:REM SPOSTA ORIGINE COLONNE VERSO SINISTRA
10 POKE36867,M2-2*X:POKE36866,M1+X:REM DIMINUISCE RIGHE E AUMENTA COLONNE
15 PRINT "R=";23-2*X;" C=";22+X
17 FORK=0TO1500:NEXTK:REM CICLO ATTESA
20 NEXTX
25 POKE36867,M2:POKE36866,M1:POKE36864,M3
```

Bit 7 : è utilizzato come bit meno significativo (LSB) del numero di linea del raggio luminoso di scansione del video, che si trova nel registro 5.

REGISTRO 5-Indirizzo 36868 (9004H)

Contenuto abituale variabile

Bit 0-7 : numero di linea su cui si trova il raggio di scansione, utilizzato insieme al bit 7 del registro 4.

REGISTRO 6-Indirizzo 36869 (9005H)

Contenuto abituale = 240

Bit 0-3 : servono per determinare l'indirizzo di inizio della mappa caratteri. Questo indirizzo è formato da 16 bit, come nello schema che segue:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	-B3	0	0	B2	B1	B0	0	0	0	0	0	0	0	0	0	0

Indirizzo Mappa Caratteri

dove B3, ecc. sono i bit 0-3 del registro. Una descrizione dettagliata del come vengono utilizzati questi bit, per costruire l'indirizzo di partenza della mappa caratteri, richiederebbe una conoscenza molto approfondita dell'hardware del calcolatore. Ci si limita quindi a precisare come essi sono interpretati. Il bit 3 (B3), negato, diventa il bit 15 dell'indirizzo, mentre i bit 0, 1 e 2 diventano rispettivamente i bit 10, 11 e 12 dell'indirizzo; i rimanenti bit dell'indirizzo sono a 0. In questo modo, se B3=0, le posizioni di partenza possibili per la mappa caratteri sono all'inizio di ogni blocco di 1K della memoria ROM, a partire dalla locazione 32768 (8000H), fino alla 39936 (9C00H), a seconda dei valori di B0, B1 e B2; se B3=1, le posizioni possibili sono ancora all'inizio di ogni blocco di 1K, ma nella memoria RAM a partire dalla locazione 0 (0000H), fino alla 7168 (1C00H).
Può essere comodo interpretare i bit 0-2 nel modo seguente: la posizione di partenza della memoria caratteri deve trovarsi all'inizio di un blocco di 1K, ed è quindi rappresentata da un numero intero di K di memoria. Si immagini di scomporre questo numero in un multiplo di 4K e in un multiplo di 1K; il bit 2 dà il valore del multiplo di 4K, che per altro può solo essere 0 o 1. I bit 0 e 1 danno il valore del multiplo di 1K, che può variare da 0 a 3.

Segue un esempio. Se si vuole collocare la mappa caratteri a partire dalla locazione 5120 in RAM, situazione molto frequente nella versione senza espansione di memoria, l'indirizzo binario è:

0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0

il bit 15 a 0 significa che B3 deve essere a 1 (negato); i bit B0, B1 e B2 ricopiano i bit

10-12 dell'indirizzo; in sostanza i bit 0-3 del registro saranno :1101.
Si sarebbe potuto anche ragionare in questi termini: la locazione 5120 inizia dopo 5K di memoria RAM, a partire da 0000; dunque B3 deve essere a 1, e inoltre, poichè 5K = 4K + 1K, B2 deve essere 1 e i bit B0 e B1 devono contenere 1.

Bit 4-7 : determinano l'indirizzo di partenza della mappa video, insieme al bit 7 del registro 3, nel modo seguente:

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	-B7	0	0	B6	B5	B4	B7 (di R3)	0	0	0	0	0	0	0	0	0

Indirizzo mappa video

Anche in questo caso, come si vede, il bit B7 viene complementato e usato come bit 15 dell'indirizzo, i bit B6, B5 e B4 del registro 6 e il bit B7 del registro 3 diventano i bit 12, 11, 10 e 9 dell'indirizzo.
La mappa video si trova a partire dalla locazione 7680 (1E00H) nella versione con espansione fino a 3K, mentre si trova a partire da 4096 (1000H) nelle altre configurazioni.
Per esempio si voglia assegnare come indirizzo di partenza della mappa video 7680.
L'indirizzo si presenta in binario nella forma:

0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0

Poichè il bit 15 è a 0, il bit B7 del registro 6 sarà a 1; gli altri bit sono anch'essi tutti a 1; quindi si avrà 1111.
Il contenuto completo del registro 6, per assegnare come indirizzo di partenza della mappa di caratteri 5120 e della mappa video 7680, sarà:

1111 1101 = FDH = 253

Come sintesi di quanto spiegato, si riporta qui di seguito una tabella che dà i diversi valori da assegnare al registro 6 per avere la mappa caratteri nelle diverse posizioni possibili, nel caso che la mappa video si trovi all'indirizzo 7680.

Indirizzo MAPPA CARATTERI			
R6	decimale	esadecimale	
248	0000	0000H	RAM Sistema
249	1024	0400H	RAM utente esp. 3K
250	2048	0800H	RAM utente esp. 3K
251	3072	0C00H	RAM utente esp. 3K
252	4096	1000H	RAM Standard
253	5120	1400H	RAM Standard
254	6144	1800H	RAM Standard
255	7168	1C00H	RAM Standard
240	32768	8000H	ROM Caratteri
241	33792	8400H	ROM Caratteri
242	34816	8800H	ROM Caratteri
243	35840	8C00H	ROM Caratteri
244	36864	9000H	ROM Sistema
245	37888	9400H	ROM Sistema
246	38912	9800H	ROM Sistema
247	39936	9C00H	ROM Sistema

Come si può vedere, nella configurazione standard del calcolatore, per spostare la mappa caratteri in RAM si può usare l'indirizzo 5120 se si vogliono usare 3K, 6144 se ne bastano 2 e 7168 se ne basta 1; naturalmente la scelta dipende anche dalla lunghezza del programma Basic, che inizia al byte 4097.
Con l'espansione da 3K, invece, dato che il Basic inizia a 1025, si potrebbero teoricamente usare anche gli indirizzi 2048, 3072 e 4096; in pratica è possibile usare solo 4096 dato che il 6561 non può collegarsi a locazioni con indirizzo minore di 4096.

Si riporta ora una tabella, analoga alla precedente, valida per espansioni superiori a 3K.

Indirizzo MAPPA CARATTERI			
R6	decimale	esadecimale	
200	0000	0000H	RAM Sistema
201	1024	0400H	RAM utente esp. 3K
202	2048	0800	RAM utente esp. 3K
203	3072	0C00H	RAM utente esp. 3K
204	4096	1000H	RAM Standard
205	5120	1400H	RAM Standard
206	6144	1800H	RAM Standard
207	7168	1C00H	RAM Standard
192	32768	8000H	ROM Caratteri
193	33792	8400H	ROM Caratteri
194	34816	8800H	ROM Caratteri
195	35840	8C00H	ROM Caratteri
196	36864	9000H	ROM Sistema
197	37888	9400H	ROM Sistema
198	38912	9800H	ROM Sistema
199	39936	9C00H	ROM Sistema

Come si può rilevare da questa tabella, se si usa l'espansione da 16K, le possibilità per la mappa dei caratteri in RAM sono poche; se il programma è corto si può usare 6144 e disporre di 2K, se il programma è più lungo si può usare solo 7168 e disporre solo di 1K. Aggiungendo più espansioni, ma in questo caso è necessario il dispositivo per collegarle contemporaneamente (cabinet), la situazione migliora e possono essere disponibili anche gli indirizzi bassi, che, come già visto, non possono essere utilizzati dal chip 6561 (essi possono servire per programmi in linguaggio macchina).
Si ricorda che il sistema vede una espansione per volta, ma, mentre collegando contemporaneamente 16K e 8K si ha sovrapposizione, questo non capita aggiungendo la 3K con una delle altre; anche se il Basic non vede la 3K, essa è presente e può essere usata per altri scopi. Naturalmente se con l'espansione da 16K o da 8K si sposta il puntatore di inizio BASIC, si possono usare per la mappa dei caratteri gli indirizzi tra 4096 e l'inizio del programma BASIC. Per costruire la tabella precedente si è usato come indirizzo della mappa video 4096, come risulta con espansioni superiori a 3K.

REGISTRO 7-Indirizzo 36870 (9006H)

contenuto abituale = 0

Contiene la coordinata orizzontale della posizione toccata dalla light pen.

REGISTRO 8-Indirizzo 36871 (9007H)

Contenuto abituale = 0

Contiene la coordinata verticale della posizione toccata dalla light pen.

REGISTRO 9-Indirizzo 36872 (9008H)

Contenuto abituale = 255

Contiene il valore di ingresso del potenziometro n. 1 del paddle.

REGISTRO 10-Indirizzo 36873 (9009H)

Contenuto abituale = 255

Contiene il valore di ingresso del potenziometro n. 2 del paddle.

REGISTRO 11-Indirizzo 36874 (900AH)

Contenuto abituale = 0

Bit 0-6 : contengono la frequenza del primo oscillatore audio.
Bit 7 : se 1 attiva l'oscillatore, se 0 lo disattiva.

REGISTRO 12-Indirizzo 36875 (900BH)

Come il registro 11, per il secondo oscillatore audio.

REGISTRO 13-Indirizzo 36876 (900CH)

Come il registro 11, per il terzo oscillatore audio.

REGISTRO 14-Indirizzo 36877 (900DH)

Come il registro 11, per il generatore di rumore bianco.

REGISTRO 15-Indirizzo 36878 (900EH)

Contenuto abituale = 0

Bit 0-3 : contengono il volume del segnale audio quando è acceso almeno un generatore di suoni.

Bit 4-7 : contengono il codice del colore ausiliario usato in modo "multicolor"; l'uso di questo colore verrà spiegato più avanti.

REGISTRO 16-Indirizzo 36879 (900FH)

Contenuto abituale = 27

Bit 0-2 : codice del colore del bordo dello schermo, uno tra gli 8 possibili.

Bit 3 : se a 1, ogni carattere viene visualizzato con il colore specificato nella mappa colori, mentre lo sfondo ha il colore definito dai bit 4-7; se a 0, tutti i caratteri vengono visualizzati con il colore comune definito dai bit 4-7, mentre lo sfondo di ogni carattere ha il colore definito nella mappa colori. In multicolor questo bit non ha effetto.

Bit 4-7 : codice di uno dei 16 colori possibili.

2.3 CARATTERI DEFINITI DALL'UTENTE

Come si è visto, la mappa caratteri standard si trova in ROM ed è quella usata dal calcolatore nel suo funzionamento ordinario. L'utente può però costruirsi una mappa caratteri personale in RAM e far sì che il sistema la usi, modificando l'indirizzo di inizio mappa caratteri nel registro 6 del 6561.
Con la tecnica dei caratteri definiti dall'utente, diventa possibile non solo creare caratteri del tutto particolari, ma anche disegnare sul video grafici di funzioni ad alta risoluzione.
Ogni carattere può essere definito usando 8 o 16 byte e si possono definire al massimo 256 caratteri diversi; nel primo caso la mappa caratteri occupa 2K, nel secondo 4K. Nella configurazione standard del VIC la memoria utente è di 3584 byte, quindi se si vogliono definire 256 caratteri diversi in RAM, questi devono essere di 8 byte; si consumano così 2048 byte e ne restano solo 1536 per il programma. Con configurazioni che comprendono espansioni di memoria RAM si può creare anche una mappa di caratteri da 16 byte ciascuno.

I caratteri che si vogliono creare devono essere descritti ponendo a 1 i bit corrispondenti ai punti che li formano e ponendo a 0 gli altri bit. La configurazione dei bit di ognuno degli 8 byte che compongono il carattere può essere assegnata o con delle POKE di costanti, introdotte conn delle DATA, in un programma Basic, o con delle istruzioni di caricamento in un programma in linguaggio macchina. Si costruiscono qui di seguito 2 caratteri particolari, che verranno usati in un programma esempio che segue.

Primo carattere:

bit		7	6	5	4	3	2	1	0	Calcolo valore byte:	
byte	0	0	0	0	1	1	0	0	0	0+0+0+16+8+0+0+0	= 24
	1	0	0	1	1	1	1	0	0	0+0+32+16+8+4+0+0	= 60
	2	0	1	1	1	1	1	1	0	0+64+32+16+8+4+2+0	= 126
	3	1	1	1	1	1	1	1	1	128+64+32+16+8+4+2+1	= 255
	4	1	1	1	1	1	1	1	1	128+64+32+16+8+4+2+1	= 255
	5	1	1	1	0	0	1	1	1	128+64+32+0+0+4+2+1	= 231
	6	1	1	0	0	0	0	1	1	128+64+0+0+0+0+2+1	= 195
	7	1	0	0	0	0	0	0	1	128+0+0+0+0+0+0+1	= 129

Secondo carattere:

Bit		7	6	5	4	3	2	1	0	Calcolo valore byte:	
Bite	0	1	1	1	0	0	0	0	0	128+64+32+0+0+0+0+0	= 224
	1	1	1	1	0	0	0	0	0	128+64+32+0+0+0+0+0	= 224
	2	1	1	1	0	0	0	0	0	128+64+32+0+0+0+0+0	= 224
	3	0	0	0	1	0	0	0	0	0+0+0+16+0+0+0+0	= 16
	4	0	0	0	0	1	0	0	0	0+0+0+0+8+0+0+0	= 8
	5	0	0	0	0	0	1	1	1	0+0+0+0+0+1+1+1	= 7
	6	0	0	0	0	0	1	1	1	0+0+0+0+0+1+1+1	= 7
	7	0	0	0	0	0	1	1	1	0+0+0+0+0+1+1+1	= 7

Ovviamente non è necessario fabbricare ex novo tutti i 256 caratteri possibili. In genere capita che l'utente voglia usare parte dei caratteri standard del VIC più un certo numero di caratteri particolari. In questo caso occorre trasferire in RAM la parte dei caratteri standard, già definiti in ROM, che si vuole usare, e aggiungere i caratteri particolari definiti. La mappa caratteri, una volta riempita, resta inalterata fino a quando si spegne il calcolatore; può quindi essere usata da più programmi.

Una volta creata in RAM la nuova mappa caratteri, si possono visualizzare i caratteri in tre modi:

.1)Scrivere il codice del carattere nella posizione desiderata della mappa video e il colore nella posizione corrispondente della mappa colori. Naturalmente il codice deve essere in DISPLAY CODE e NON in ASCII.

.2)Scrivere il carattere con una PRINT, usando la funzione CHR$, mettendo tra parentesi il codice del carattere ASCII che corrisponde al carattere scelto.

.3)Scrivere il carattere con una PRINT, ponendo tra le virgolette una stringa formata dal carattere corrispondente al codice nella mappa standard. Se il display code usato supera 128, nella stringa si deve far entrare il carattere RVS-ON, prima del carattere da stampare, e il carattere RVS-OFF, dopo.

Quanto detto potrà sembrare complicato, ma fa capire meglio come funziona il calcolatore. I caratteri stampabili sono solo 128; precisamente 128 diretti e 128 inversi. I caratteri vengono stampati come inversi in base al valore del bit 3 del registro 16 del 6561. Quando si scrive direttamente con POKE nella mappa video si possono usare anche codici superiori a 128; infatti il numero scritto viene usato solo come puntatore.

Segue un programma che stampa sul video, e volendo anche sulla stampante, tutti i caratteri stampabili, con i codici ASCII e i DISPLAY CODE, per i seguenti intervalli dei codici ASCII:

.da 32 a 127
.da 160 a 255.

```
1 REM X7
2 PRINT"♥";
3 DIMA(96),D(96),C$(96):REM VETTORI DI LAVORO
4 REM DEFINIZIONE INDIRIZZI MAPPA VIDEO E MAPPA COLORI
5 IFPEEK(648)=30THENMV=7680:MC=38400:GOTO8
7 MV=4096:MC=37888
8 PRINT"♥VUOI USARE LA PRINTER S/N?":INPUTR$
10 REM STAMPA CODICI ASCII DA 32 A 63
15 K1=32:K2=63:GOSUB1000
20 GOSUB1100
25 GOSUB1200
30 GOSUB1300
40 REM STAMPA CODICI DA 64 A 127
45 K1=64:K2=127:GOSUB1000
50 GOSUB1100:GOSUB1200:GOSUB1300
55 REM STAMPA CODICI DA 160 A 255
60 K1=160:K2=255:GOSUB1000
65 GOSUB1100:GOSUB1200:GOSUB1300
999 STOP
1000 PRINT"♥";:FORK=K1TOK2:PRINTCHR$(K);:NEXTK:PRINT:RETURN
1100 I=1:FORK=K1TOK2:A(I)=K:D(I)=PEEK(MV-1+I):C$(I)=CHR$(K):I=I+1:NEXTK:RETURN
1200 I=1:FORK=K1TOK2:PRINTA(I);"     ";D(I);"     ";C$(I)
1210 FORJ=0TO500:NEXTJ:I=I+1:NEXTK:RETURN
1300 IFR$="N"THENRETURN
1305 OPEN4,4:CMD4:PRINT"CORRISPONDENZA CODICI ASCII, DISPLAY CODE E CARATTERI"
1310 PRINT:PRINT"ASCII  D/CODE  CARATT.":PRINT
1315 GOSUB1200:PRINT#4:CLOSE4:RETURN
```

Il programma funziona con qualunque configurazione di memoria: infatti alle linee 5 e 7 vengono definiti i puntatori per il cambio di indirizzi delle mappe video e colori (in realtà questo ultimo indirizzo non è necessario, ma viene scritto per ricordare come operare in programmi analoghi, che stampano sul video senza usare le PRINT). Alla linea 3 vengono dimensionati 3 vettori: A, D e C$, per contenere rispettivamente: codici ASCII, DISPLAY CODE e CARATTERI come stringhe. I vettori vengono riempiti e stampati in tre fasi, per codici da 32 a 63, per codici da 64 a 127 e per codici da 160 a 255. Il sottoprogramma che inizia alla locazione 1000 pulisce il video e stampa con una PRINT, nella prima posizione del video, il carattere, usando la funzione CHR$. Il sottoprogramma che si trova alla linea 1100 memorizza il codice ASCII in A(I), il DISPLAY CODE, ottenuto con la funzione PEEK sulla prima posizione del video, in D(I) e il carattere ottenuto con la funzione CHR$ in C$(I). Il sottoprogramma alla linea 1200 stampa i risultati sul video, e, se si è risposto affermativamente alla domanda iniziale sull'uso della stampante, anche sulla stampante.
Seguono i risultati del programma precedente.

CORRISPONDENZA CODICI ASCII, DISPLAY CODE E CARATTERI

ASCII	D/CODE	CARATT.	ASCII	D/CODE	CARATT.	ASCII	D/CODE	CARATT.
32	32		64	0	@	96	64	─
33	33	!	65	1	A	97	65	♠
34	34	"	66	2	B	98	66	│
35	35	#	67	3	C	99	67	─
36	36	$	68	4	D	100	68	─
37	37	%	69	5	E	101	69	─
38	38	&	70	6	F	102	70	─
39	39	'	71	7	G	103	71	│
40	40	(	72	8	H	104	72	│
41	41	)	73	9	I	105	73	╮
42	42	*	74	10	J	106	74	╰
43	43	+	75	11	K	107	75	╯
44	44	,	76	12	L	108	76	L
45	45	-	77	13	M	109	77	╲
46	46	.	78	14	N	110	78	╱
47	47	/	79	15	O	111	79	┌
48	48	0	80	16	P	112	80	┐
49	49	1	81	17	Q	113	81	●
50	50	2	82	18	R	114	82	─
51	51	3	83	19	S	115	83	♥
52	52	4	84	20	T	116	84	│
53	53	5	85	21	U	117	85	╭
54	54	6	86	22	V	118	86	╳
55	55	7	87	23	W	119	87	○
56	56	8	88	24	X	120	88	♣
57	57	9	89	25	Y	121	89	│
58	58	:	90	26	Z	122	90	♦
59	59	;	91	27	[	123	91	┼
60	60	<	92	28	£	124	92	▒
61	61	=	93	29	]	125	93	│
62	62	>	94	30	↑	126	94	π
63	63	?	95	31	←	127	95	◥

CORRISPONDENZA CODICI ASCII, DISPLAY CODE E CARATTERI

ASCII	D/CODE	CARATT.	ASCII	D/CODE	CARATT.	ASCII	D/CODE	CARATT.
160	96		192	64	-	224	96	
161	97	▮	193	65	♠	225	97	▮
162	98	■	194	66	|	226	98	■
163	99	‾	195	67	-	227	99	‾
164	100	_	196	68	-	228	100	_
165	101	|	197	69	-	229	101	|
166	102	▒	198	70	_	230	102	▒
167	103	|	199	71	|	231	103	|
168	104	▪	200	72	|	232	104	▪
169	105	◤	201	73	╲	233	105	◤
170	106	|	202	74	╲	234	106	|
171	107	├	203	75	╯	235	107	├
172	108	▪	204	76	L	236	108	▪
173	109	└	205	77	\	237	109	└
174	110	┐	206	78	/	238	110	┐
175	111	-	207	79	┌	239	111	-
176	112	┌	208	80	┐	240	112	┌
177	113	┴	209	81	●	241	113	┴
178	114	┬	210	82	-	242	114	┬
179	115	┤	211	83	♦	243	115	┤
180	116	|	212	84	|	244	116	|
181	117	▮	213	85	/	245	117	▮
182	118	▮	214	86	×	246	118	▮
183	119	‾	215	87	O	247	119	‾
184	120	▬	216	88	♣	248	120	▬
185	121	▬	217	89	|	249	121	▬
186	122	┘	218	90	♦	250	122	┘
187	123	▪	219	91	+	251	123	▪
188	124	▪	220	92	▒	252	124	▪
189	125	┘	221	93	|	253	125	┘
190	126	▪	222	94	π	254	126	▪
191	127	▚	223	95	◥	255	94	π

Si riportano le formule che consentono di passare da DISPLAY CODE (D) a codice ASCII (A):

$0 \leqslant D \leqslant 31$	A=D+64
$32 \leqslant D \leqslant 63$	A=D
$64 \leqslant D \leqslant 95$	A=D+32
$96 \leqslant D \leqslant 127$	A=D+64

A questo punto si può esporre il listato del programma che serve per costruire con qualunque configurazione di memoria una mappa caratteri in RAM ricopiando 128 caratteri dalla mappa della ROM (i primi), andando a sostituire ad alcuni di questi i due caratteri precedentemente costruiti per punti e aggiungendoli anche dopo i primi 128. In sostanza si assegnano a ciascuno di questi 2 caratteri 4 codici diversi.
Il programma inizia con due linee, la 10 e la 13, dove vengono sistemati, in base alla

```
1 REM X6
10 L=253:M=5120:MV=7680:MC=38400:IFPEEK(648)=30THEN17
13 MV=4096:MC=37888:M=6144:L=206
17 FORJ=0TO1023:POKEM+J,PEEK(32768+J):NEXTJ
20 FORJ=0TO15:READA
25 X=M+J:Y=X+61*8:POKEY,A
26 Y=X+163*8:POKEY,A
27 Y=X+91*8:POKEY,A
28 Y=X+123*8:POKEY,A
30 NEXTJ
35 POKE36869,L
55 DATA24,60,126,255,255,231,195,129
60 DATA224,224,224,16,8,7,7,7
120 PRINT"🗏STAMPA 4 VOLTE I 2 CARATTERI GENERATI, IN 3 MODI DIVERSI:"
121 PRINTCHR$(61);CHR$(62);"█";CHR$(163-128);CHR$(164-128);"█";
122 PRINTCHR$(91+32);CHR$(92+32);CHR$(123+64);CHR$(124+64)
123 PRINT"=";">";"█*";"$█";"+";"*";"▪";" ▀"
126 X=MV+9*22:Y=MC+9*22:POKEX,61:POKEY,2
127 X=MV+9*22+11:Y=MC+9*22+11:POKEX,62:POKEY,2
128 X=MV+10*22:Y=MC+10*22:POKEX,163:POKEY,4
129 X=MV+10*22+11:Y=MC+10*22+11:POKEX,164:POKEY,4
130 X=MV+11*22:Y=MC+11*22:POKEX,91:POKEY,6
131 X=MV+11*22+11:Y=MC+11*22+11:POKEX,92:POKEY,6
132 X=MV+12*22:Y=MC+12*22:POKEX,123:POKEY,5
133 X=MV+12*22+11:Y=MC+12*22+11:POKEX,124:POKEY,5
150 STOP
```

posizione della mappa video, i seguenti parametri:

.L per sistemare il byte 36869 e quindi puntare alla mappa caratteri voluta;
.M per definire l'indirizzo di inizio della mappa caratteri;
.MV per definire la posizione della mappa video;
.MC per definire la posizione della mappa colori.

Alla linea 10 vengono trasferiti nella mappa caratteri in RAM i primi 128 caratteri della mappa in ROM. Dalla 20 alla 30 vengono trasferiti al posto dei display code 61 e 62, 163 e 164, 91 e 92, 123 e 124, i due caratteri definiti nelle DATA delle linee 55 e 60. Gli indirizzi della mappa caratteri sono stati scelti in modo di poter disporre di 2K. Alle linee 121 e 122 vengono stampate sul video le 4 coppie dei caratteri personali usando la funzione CHR$. Alla linea 123 si fa la stessa cosa usando le stringhe dopo la PRINT. Nelle linee da 126 a 133 si usano invece le POKE nella mappa video e nella mappa colore per ottenere i caratteri.

Si deve scegliere opportunamente l'indirizzo della mappa caratteri in RAM per non sovrapporsi al programma BASIC. Nell'esempio si è evitato l'uso di commenti per non allungare il programma.

Segue un programma scritto per configurazione standard o con espansione fino a 3K. Esso, dopo aver spostato il cursore verso il basso del video, alla linea 5, introduce nella mappa caratteri, che inizia in 5120, 32 volte il carattere A, 32 volte il carattere B e 64 volte il carattere C, 32 volte il carattere D, 32 volte il carattere E e 64

volte il carattere F. Alla linea 50 viene modificato il byte 36869 per puntare alla mappa caratteri in 5120. Alla linea 60 viene controllato che il primo bit del registro 36866 sia 1. Alla linea 80 vengono introdotti con delle POKE i numeri da 0 a 255 nella mappa video (il 7680) e il codice colore 2 (rosso) in tutte le corrispondenti posizioni della mappa colori. Sul video appaiono in rosso 32 A, 32 B, 64 C, 32 D, 32 E e 64 F. Al posto del solito READY, si vedrà AAAAAB. Premendo contemporaneamente RUN-STOP e RESTORE tutto scompare e si torna alle condizioni normali di accensione.

```
1 REM X8
2 REM METTE NELLA MAPPA VIDEO 32 A, 32 B, 64 C, 32 D, 32 E, 64 F
3 REM TRASFERISCE QUESTI CARATTERI NELLA RAM A PARTIRE DA 5120
4 REM PORTA I CARATTERI SUL VIDEO CON POKE, METTENDO NUMERI DA 0 A 255
5 REM DA PROVARE SENZA ESPANSIONI DI MEMORIA O CON ESP. DA 3K
6 PRINT"]QQQQQQQQQQQQQQQQQQQ";
10 K1=0:K2=31:D=1:GOSUB150
15 K1=32:K2=63:D=2:GOSUB150
20 K1=64:K2=127:D=3:GOSUB150
25 K1=0:K2=31:D=4:GOSUB200
30 K1=32:K2=63:D=5:GOSUB200
35 K1=64:K2=127:D=6:GOSUB200
50 POKE36869,253
60 POKE36866,PEEK(36866)OR128
80 FORK=0TO255:POKE7680+K,K:POKE38400+K,2:NEXTK
100 END
150 FORK=K1TOK2:FORJ=0TO8:POKE(5120+K*8+J),PEEK(32768+D*8+J):NEXTJ:NEXTK:RETURN
200 FORK=K1TOK2:FORJ=0TO8:POKE(6144+K*8+J),PEEK(32768+D*8+J):NEXTJ:NEXTK:RETURN
```

Qualora si voglia usare il programma con espansioni superiori ai 3K, si devono modificare opportunamente le linee 50, 60, 80, 150 e 200.

Segue un programma che usa il formato dei caratteri 16x8. Esso definisce un carattere A allungato nella mappa di caratteri in RAM e lo stampa sul video 256 volte.

```
1 REM X9
2 REM PROVA RIEMPIMENTO VIDEO 255 A INGRANDITE
3 REM NEL FORMATO 16X8
4 REM OGNI CARATTERE OCCUPA 2 RIGHE
5 PRINT"]"
10 FORK=0TO15:READA%:POKE(6144+K),A%:NEXTK
20 MV=7680:MC=38400:L=254:R=240:S=150
30 IFPEEK(648)=16THENMV=4096:MC=37888:L=206:R=192:S=22
50 POKE36869,L
51 POKE36867,PEEK(36867)OR1
80 FORK=0TO255:POKEMV+K,0:POKEMC+K,2:NEXTK
100 GETA$:IFA$=""THEN100
101 POKE36866,S:POKE36867,174
102 POKE36869,R:FORK=0TO255:POKEMV+K,32:NEXTK
103 END
110 DATA24,24,36,36,66,66,66,66
120 DATA126,126,66,66,66,66,0,0
```

Il carattere è stato definito così:

Primo carattere	0 0 0 1 1 0 0 0 = 24
	0 0 0 1 1 0 0 0 = 24
	0 0 1 0 0 1 0 0 = 36
	0 0 1 0 0 1 0 0 = 36
	0 1 0 0 0 0 1 0 = 66
	0 1 0 0 0 0 1 0 = 66
	0 1 0 0 0 0 1 0 = 66
	0 1 0 0 0 0 1 0 = 66
Secondo carattere	0 1 1 1 1 1 1 0 = 126
	0 1 1 1 1 1 1 0 = 126
	0 1 0 0 0 0 1 0 = 66
	0 1 0 0 0 0 1 0 = 66
	0 1 0 0 0 0 1 0 = 66
	0 1 0 0 0 0 1 0 = 66
	0 0 0 0 0 0 0 0 = 0
	0 0 0 0 0 0 0 0 = 0

Il programma funziona con qualunque espansione; infatti alle linee 20 e 30 sistema i puntatori per accedere alle mappe. Viene costruito il carattere A nei primi 16 byte della mappa caratteri, che inizia in 6144. Alla linea 50 viene messo in 36869 il puntatore alla mappa. Alla linea 51 viene posto a 1 il bit 0 del registro 36867 che controlla il modo 16x8. Alla linea 80 si azzera il contenuto della mappa video, per puntare sempre allo stesso carattere A, individuato dal display code 0. Sempre alla linea 80 vengono posti 256 codici colore 2 (rosso) nella mappa colori. Alla linea 100 il programma si mette in attesa della pressione di un tasto per uscire e ripristinare la situazione normale del video e dei caratteri.

2.4 GRAFICA AD ALTA RISOLUZIONE

Nelle applicazioni di grafica ad alta risoluzione il video viene riempito con tutti i 256 caratteri definiti nella mappa caratteri e quest'ultima diventa un'immagine del video in memoria, punto per punto. La mappa video viene riempita, a partire dall'inizio, con codici crescenti da 0 a 255, mentre in ogni carattere della mappa caratteri vengono messi a 1 i punti che devono costituire il disegno o il grafico. Se i caratteri sono descritti con 8 byte, si copre circa la metà dello schermo; infatti i caratteri visualizzabili sono 256, mentre le posizioni del video sono 506. Per poter utilizzare tutto il video, ogni carattere deve occupare 16 byte, cioè due posizioni

carattere sul video in verticale; per ottenere questo basta porre a 1 il bit 0 del registro 4 del 6561, ricordando però che in questo caso la mappa caratteri va ad occupare 4K invece di 2K.
In genere, un programma per produrre grafici ad alta risoluzione contiene una parte di inizializzazione e una di stampa del grafico. Nella prima parte vengono assegnati opportuni valori ai registri del 6561, per poter utilizzare la mappa caratteri definita dall'utente; inoltre vengono riempite la mappa video e la mappa colori , e viene ripulita, azzerandola, la mappa caratteri. Nella seconda parte viene costruito e visualizzato il grafico, usando un algoritmo che calcoli, in base alle coordinate X e Y di un punto, la posizione del carattere coinvolto, e, al suo interno, la posizione del bit che deve essere posto al valore 1.

Segue un esempio; il programma traccia il grafico della funzione:

y=45+40 cos(x/10)

per x che varia da 0 a 175. Il programma funziona con qualunque configurazione di memoria e usa la mappa caratteri a partire da 5120 o da 6144. Dato che non sono disponibili più di 2K, si è dovuto dimezzare il quadro video, spostando in giù l'origine dello stesso e riducendo a 11 le righe. Le colonne sono state mantenute a 22. Il grafico può occupare 176x88 punti, e la funzione è stata scelta in modo da non superare tali valori. Il grafico si presenta ribaltato, con l'asse x orizzontale, rivolta verso destra, in alto sul video, e l'asse y verticale, rivolta verso il basso, sulla sinistra. Operando così non si è dovuto ricorrere a fattori di scala per far rientrare il grafico nel quadro video. Alla linea 10 viene ridotto a 0 il numero di righe del quadro e viene spostata in giù l'origine. Alla 15 vengono preparati in F(I) i punti del grafico nell'ambito di una posizione carattere. Alla 20 viene riempita la mappa video con numeri da 0 a 255 (in realtà ne bastano meno, dato che i caratteri disponibili sono 22x11=242), e la mappa colori con il codice del rosso. Alla 30 vengono messi a zero tutti i bit della mappa caratteri. Alla 40 viene predisposto il puntatore alla mappa caratteri e vengono definite 11 righe per il quadro video. Da 80 a 112 si sviluppa l'algoritmo di calcolo per punti della funzione. Dopo aver calcolato il valore della y in un punto, viene calcolata la quota (riga) alla quale si trova il carattere corrispondente; poi, usando il resto, ottenuto con la funzione INT, si trova quale bit nella posizione carattere deve essere posto a 1. Anche questo programma termina mettendosi in attesa della pressione di un tasto. Si usa tale accorgimento per evitare di rovinare il quadro video. Dopo la pressione di un qualunque tasto vengono ripristinate le condizioni iniziali del calcolatore.

```
1 REM GRAFICO1
7 PRINT"■♥"
8 MV=7680:MC=38400:L=253:L1=240:L2=150:CV=5120
9 IFPEEK(648)=16THENMV=4096:MC=37888:L=206:L1=192:L2=22:CV=6144
10 POKE36867,128:POKE36865,60
15 F(0)=128:F(1)=64:F(2)=32:F(3)=16:F(4)=8:F(5)=4:F(6)=2:F(7)=1:F(8)=0
20 FORK=0TO255:POKEMV+K,K:POKEMC+K,2:NEXTK
30 FORK=CVTOCV+255*8:POKEK,0:NEXTK
40 POKE36869,L:POKE36867,150
80 FORX=0TO175
90 Y=45+40*COS(X/10)
101 I=CV
102 Q=Y/8
103 IQ=INT(Q)
104 I=I+(IQ*176)
105 Q=(Q-IQ)*8
106 R=X/8
107 IR=INT(R)
108 I=I+(IR*8)
109 I=I+Q
110 R=INT((R-IR)*8)
111 POKEI,PEEK(I)ORF(R)
112 NEXTX
1003 GETA$:IFA$=""THEN1003
1004 FORK=0TO255:POKEMV+K,32:NEXTK
1005 POKE36866,L2
1006 POKE36867,174
1007 POKE36865,38
1008 POKE36869,L1
1009 END
```

Segue un programma che traccia il grafico della funzione X*SIN(X), usando il formato 16x8. Per far girare il programma si deve spostare l'inizio del BASIC e lasciare liberi 4K da 4096 a 8192 (1000H-2000H). Il programma necessita di espansione di memoria da 16K o da 8K. I 4K lasciati liberi a partire da 4096 sono così utilizzati:

.da 4096 a 4351, 256 byte per la mappa video;

.da 4352 a 8191, 3840 byte per la mappa dei caratteri, cioè per costruire per punti la funzione. In realtà la funzione usa 3696 punti (21x11x2x8=3696), e quindi restano inutilizzati un certo numero di byte. Per costruire la funzione si deve usare un algoritmo che tenga conto del modo nel quale i punti vanno poi a posizionarsi sul video; si ricorda che i primi 8 byte della mappa caratteri vanno nella prima posizione carattere della prima riga del video, mentre gli 8 byte seguenti vanno nella prima posizione carattere della seconda riga.

```
1 REM GRAFICO2
20 REM   ATTENZIONE : BATTUTO IL PROGRAMMA SALVARLO E, PRIMA DI RICARICARLO,
30 REM   ==========   DARE IL SEGUENTE ORDINE : POKE642,32:SYS58232 .
40 REM                CIO' PERMETTE AL VIC DI RILOCARE IL PROGRAMMA PARTENDO
50 REM                DALLA LOCAZIONE 8192 ($2000) LASCIANDO COSI' LIBERA LA
60 REM                ZONA DI MEMORIA COMPRESA TRA 4096 E 8192 ($1000-$2000).
70 REM                QUESTA ZONA VERRA' UTILIZZATA PER LA MAPPA VIDEO E LA
80 REM -              MAPPA DEI CARATTERI.
100 PRINT"■■■■■■DIMOSTAZIONE  USO    ■MATRICI CARATTERI          ■16 * 8"
110 PRINT"■■■■■■ATTENDERE PREGO"
120 DEFFNY(X)=SIN(X)*X
130 FORI=4352TO8191:POKEI,0:NEXT:REM  SBIANCA I CARATTERI
140 FORI=0TO20:FORJ=0TO10:POKE4096+I+21*J,J+11*I+16:POKE37888+I+21*J,1:NEXTJ,I
150 REM RIEMPE IL VIDEO DI CARATTERI
160 POKE36866,21:REM NUMERO COLONNE = 21
170 POKE36867,151:REM NUMERO RIGHE = 11 ; MATRICI = 16X8
180 POKE36869,204:REM INIZIO MAPPA CARATTERI A 4096
200 REM USEREMO LA MAPPA DEI CARATTERI A PARTIRE DA 4352 E NON DA 4096 PER
210 REM NON SOVRAPPORLA CON LA MAPPA VIDEO (256 BYTE)
220 REM IL PRIMO CODICE CARATTERE USATO E' PERTANTO 16
230 POKE36879,14:REM COLORE
240 FORX=0TO167:Y=FNY(X*9*π/168-9*π/2)*168/(9*π)+84:GOSUB320
250 REM DISEGNA LA FUNZIONE E L'ASSE DELLE X
260 Y=84:GOSUB320:IFX=84THENGOSUB360
270 NEXT
280 GOTO280:REM ATTENDE STOP RESTORE
300 REM  ROUTINE DI PLOTTAGGIO DEL PUNTO NELLA MAPPA DEI CARATTERI
320 DX=INT(X/8):CE=4352+176*DX+176-Y:NR=PEEK(CE)OR2↑INT(7-(X/8-DX)*8+.5):POKECE,
NR:RETURN
340 REM  DISEGNA L'ASSE DELLE Y
360 FORY=1TO176:GOSUB320:NEXT:RETURN
```

Il programma reca nelle REM iniziali le spiegazioni su quello che si deve fare prima di caricarlo per spostare l'inizio del BASIC. La funzione da calcolare viene definita alla linea 120. Essa viene studiata nell'intervallo -(9/2)*PI e +(9/2)*PI; l'intervallo viene adattato con il fattore di scala ai 168 punti disponibili orizzontalmente e ai 176 disponibili verticalmente. La mappa video inizia a 4096 e la mappa colori a 37888. Alla linea 130 vengono "blankati" i caratteri della mappa video e viene posto 1 nelle posizioni della mappa colore.
La prima riga del video corrisponde ai byte della mappa che vanno da 4096 a 4116 (21 byte); in realtà, ogni posizione carattere verrà allungata e andrà ad occupare 2 righe del video (una colonna di 16 byte).La seconda riga del video (che sarà poi la terza + la quarta) corrisponde ai byte che vanno da 4117 a 4137, ecc. Alla linea 140 vengono scritti nei byte della mappa video dei numeri crescenti, a partire da 16; essi sono i puntatori alla mappa caratteri, dove viene costruita per punti la funzione. Si parte da 16 e si arriva a 26 sulla prima colonna, da 27 a 37 sulla seconda ecc. La ragione per cui si parte da 16, invece che da 0 o 1 è che per il sistema la mappa caratteri può partire a blocchi fissi, e qui parte da 4096; ma 4096 è anche l'inizio della mappa video, quindi è necessario saltare 256 byte (16 caratteri occupano

16x16 byte) per non avere sovrapposizione. Alla linea 20 viene calcolato il valore della funzione in ognuno dei 168 punti. Per ogni punto calcolato, il sottoprogramma alla linea 320 calcola in quale posizione della mappa caratteri deve essere segnato un punto. Vengono disegnati gli assi, x orizzontale e y verticale.
Per interrompere occorre premere RUN-STOP RESTORE.

2.5 COLORE

Il colore delle diverse posizioni del video è controllato dai registri 15 e 16 del 6561 e dalla mappa colori. Quest'ultima è una zona di memoria RAM di 506 byte, situata all'indirizzo 38400 (9600H) per configurazione standard e con 3K di espansione, e all'indirizzo 37888 (9400H) per espansioni superiori a 3K. Di ogni byte della mappa colori vengono usati solo i 4 bit meno significativi (bit 0-3).
Esistono 2 modi per lavorare con il colore: il modo ALTA RISOLUZIONE e il modo MULTICOLOR, che possono essere selezionati in modo indipendente per le diverse posizioni del video. La tecnica "ad alta risoluzione" per quanto riguarda il colore non ha niente a che vedere con la "grafica ad alta risoluzione" studiata nel precedente paragrafo. Può sembrare poco felice la scelta di una terminologia identica per due applicazioni differenti; d'altra parte questa terminologia è stata adottata nei manuali e nella documentazione originale Commodore. Una probabile giustificazione sta nel fatto che, come la grafica ad alta risoluzione fornisce la possibilità di raddoppiare la risoluzione verticale con il formato 16x8, così la tecnica del colore ad alta risoluzione consente una risoluzione orizzontale doppia rispetto al multicolor.

2.5.1 ALTA RISOLUZIONE

Il modo normale di lavoro è quello ad alta risoluzione, che è selezionato quando il bit 3 del byte della mappa colori è a 0. In questo modo ad ogni bit 1 del carattere in questione corrisponde un punto sul video. Ad ogni carattere devono essere assegnati 2 colori, uno per i punti che lo compongono (bit a 1), e uno per i punti dello sfondo (bit a 0). Uno dei due colori è selezionato dai bit 4-7 del registro 16, l'altro dai bit 0-2 del corrispondente byte della mappa colori. Se il bit 3 del registro 16 è a 1, i caratteri compaiono con il colore definito nella mappa colori e con il colore di sfondo definito nel registro 16 stesso, se invece il bit 3 è a 0, i colori vengono invertiti, cioè i caratteri acquistano tutti lo stesso colore selezionato dal registro 16, su sfondi di colore diverso, definito in ogni byte della mappa colori. Infine i bit 0-2 del registro 16 selezionano il colore del bordo dello schermo.

Il colore selezionato dai bit 4-7 del registro 16 può essere uno dei seguenti:

Codice	Colore	Codice	Colore
0	nero	8	arancione
1	bianco	9	arancione chiaro
2	rosso	10	rosa
3	viola	11	viola chiaro
4	magenta	12	magenta chiaro
5	verde	13	verde chiaro
6	blu	14	blu chiaro
7	giallo	15	giallo chiaro

Il colore selezionato dai bit 0-2 della mappa colori e dai bit 0-2 del registro 16 può essere solo uno dei primi 8 della precedente tabella.

Al momento dell'accensione, la configurazione del registro 16 e dei byte della mappa colori seleziona il bianco, sia per lo sfondo che per il carattere, e il magenta per il bordo; per visualizzare un carattere occorre naturalmente assegnare due colori diversi per il carattere e per lo sfondo. Seguono alcuni esempi di selezione colore.

```
POKE 36879,PEEK(36879) AND247
```

mette a 0 il bit 3 del registro 16; i caratteri avranno tutti lo stesso colore su sfondi diversi dipendenti dal contenuto del byte nella mappa colori (COMMON FORE-GROUND).

```
POKE 36879,PEEK(36879) OR8
```

mette a 1 il bit 3 del registro 16; i caratteri avranno colori diversi dipendenti dal contenuto del byte nella mappa colori (COMMON BACKGROUND).

```
POKE 36879,PEEK(36879) AND15
POKE 36879,PEEK(36879) OR(10*16)
```

la prima istruzione azzera i bit 4-7 del registro 16; la seconda ci scrive il colore rosa (cod. 10). La moltiplicazione per 16 (2 elevato alla 4) rappresenta uno spostamento del codice di 4 posizioni a sinistra, in modo da inserirlo nei 4 bit più significativi del registro.

```
POKE 36879,PEEK(36879) AND248
POKE 36879,PEEK(36879) OR7
```

la prima istruzione azzera i bit 0-2 del registro 16, destinati al colore del bordo, la seconda ci scrive il colore giallo (cod. 7).

```
10 P = 22*Y+X
20 POKE 38400+P,C
30 POKE 7680+P,A
```

P è la posizione del video che si trova all'incrocio della colonna X e della riga Y; C è il codice del colore da scrivere nella mappa colori e A è il codice del carattere in display code da scrivere nella mappa video. L'esempio si riferisce ad una configurazione del calcolatore fino a 3K di espansione.

Segue un programma che stampa sul video i semi delle carte da gioco e fa variare i colori modificando nel registro 16 i bit relativi al colore dello sfondo, del bordo e del modo diretto/inverso. I caratteri vengono evidenziati introducendo con delle POKE i codici nella mappa video, e analogamente variando nelle posizioni corrispondenti della mappa colori i codici del colore. Il programma funziona con qualunque configurazione di memoria.

```
1 REM COLORI
2 REM PROVA COLORI ALTA RISOLUZIONE
10 MV=7680:MC=38400
15 IFPEEK(640)=16THENMV=4096:MC=37888
17 PRINT"♥";
20 POKEMV+25,65:POKEMV+30,83:POKEMV+35,90:POKEMV+40,88
30 POKEMV+69,65:POKEMV+74,83:POKEMV+79,90:POKEMV+84,88
35 X=0
40 FORK=0TO15
45 POKE36879,(PEEK(36879)AND15)ORK*2↑4
50 FORI=0TO7
55 POKE36879,(PEEK(36879)AND248)ORI
60 POKEMC+25,I:POKEMC+30,I:POKEMC+35,I:POKEMC+40,I
65 POKEMC+69,I:POKEMC+74,I:POKEMC+79,I:POKEMC+84,I
70 FORJ=1TO300:NEXTJ
75 NEXTI
77 IFX=0THENX=1:GOTO79
78 X=0
79 POKE36879,(PEEK(36879)AND247)ORX*2↑3
80 NEXTK
90 END
```

Segue un programma che produce sul video l'effetto di un caleidoscopio. Esso funziona con qualunque configurazione di memoria. La tecnica usata consiste nel riempire la mappa video con il codice 160 dello spazio inverso. Poi, partendo dalle posizioni centrali del video, si lavora sulla mappa colori e si modificano in modo simmetrico i codici colore nelle posizioni circostanti la zona centrale. I codici colore variano estraendo numeri a caso compresi tra 0 e 15. Il programma gira in continuazione; per interromperlo premere RUN-STOP e RESTORE insieme.

```
1 REM CALEIDOSCOPIO
5 REM SCELTA CONFIGURAZIONE
10 MV=7680:MC=38400
15 REM RIEMPIE LA MAPPA VIDEO DI SPAZI INVERSI
20 IFPEEK(648)=16THENMV=4096:MC=37888
25 PRINT"♥";
40 FORK=0TO505:POKE(MV+K),160:NEXTK
45 Z=MC+253:L=0
50 FORI=1TO9
55 FORJ=0TOL
63 K=(15*RND(0))+1
65 POKE(Z+J),K
67 POKE(Z-J-1),K
70 POKE(Z+22-21*I+22*J),K
75 POKE(Z-21*I-22*J),K
80 POKE(Z-23*I+21+22*J),K
85 POKE(Z-23*I-1-22*J),K
90 POKE(Z-44*I+21-J),K
95 POKE(Z-44*I+22+J),K
100 NEXTJ
105 L=L+1:Z=Z+22
110 NEXTI
120 GOTO45
```

2.5.2 MULTICOLOR

In questo modo ogni carattere della mappa caratteri, descritto come al solito in 8 byte, viene visto come una matrice di 4x8 colori, con una corrispondenza cioè di 2 bit colore per ogni coppia di punti che compare sul video. Questa tecnica riduce la risoluzione dell'immagine, ma consente di creare caratteri variopinti. Per poter apprezzare in pieno questa caratteristica del VIC bisogna disporre di un televisore a colori di buona qualità.
Nella figura 2.3 è schematizzata la corrispondenza tra bit e punti in multicolor. lor.

I due bit che rappresentano due punti indicano un codice che va da 00 a 11 che è il codice colore dei due punti e può essere scelto tra quelli selezionati nei registri 15 e

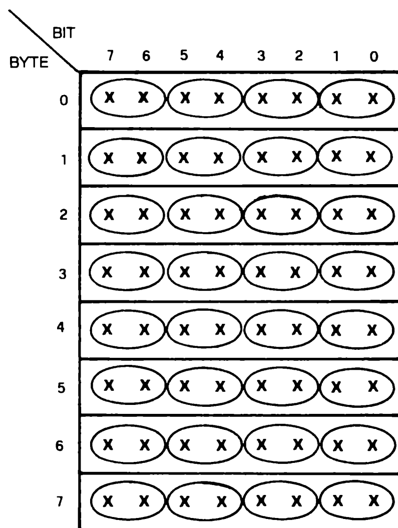


Figura 2.3 Corrispondenza bit/punti in multicolor

16 e nel corrispondente byte della mappa colori. Precisamente ogni codice seleziona i seguenti colori:

00:bit 4-7 registro 16 (colore sfondo)

01:bit 0-2 registro 16 (colore bordo)

10:bit 0-2 byte corrispondente mappa colori (colore carattere)

11:bit 4-7 registro 15 (colore ausiliario)

Un carattere definito nella mappa caratteri come segue:

bit		7	6	5	4	3	2	1	0
byte	0	0	0	0	1	1	0	1	1
	1	0	0	0	1	1	0	1	1
	2	0	0	0	1	1	0	1	1
	3	0	0	0	1	1	0	1	1
	4	1	1	1	0	0	1	0	0
	5	1	1	1	0	0	1	0	0
	6	1	1	1	0	0	1	0	0
	7	1	1	1	0	0	1	0	0

comparirà sul video così:

AABBCCDD
AABBCCDD
AABBCCDD
AABBCCDD
DDCCBBAA
DDCCBBAA
DDCCBBAA
DDCCBBAA

occupando lo stesso spazio occupato normalmente da un carattere in senso orizzontale.
In sostanza, mentre in alta risoluzione in ogni posizione del video possono comparire 2 colori, uno per il carattere e uno per lo sfondo, in modo multicolor ne possono comparire fino a 4.

Il modo multicolor consente una grande flessibilità nell'uso dei colori, quando si lavora con una mappa caratteri in RAM, definita dall'utente, mentre, in linea di massima, non ha molto significato usarlo con la mappa caratteri standard residente in ROM, i cui caratteri sono previsti per il modo ad alta risoluzione, e, visualizzati in multicolor, apparirebbero come una matrice di punti variopinti e disordinati piuttosto che come un carattere.

Dal momento che il modo colore è selezionato carattere per carattere dal bit 3 del byte corrispondente nella mappa colori, in una stessa immagine possono comparire caratteri ad alta risoluzione e caratteri in multicolor. Ogni carattere, come si è visto, occupa lo stesso spazio, sia in memoria che sul video, anche se i caratteri in multicolor hanno un potere risolutivo per numero di punti in senso orizzontale pari alla metà degli altri.

Il bit 3 del registro 16 non ha effetto in multicolor, per ovvi motivi; se però in una stessa immagine vengono usati i due modi, gli deve essere assegnato un valore opportuno per i caratteri ad alta risoluzione.

Riassumendo, le operazioni necessarie per operare in multicolor sono:

.1)assegnare un colore nei bit 4-7 del registro 16;

.2)assegnare un colore al bordo nei bit 0-2 del registro 16;

.3)assegnare un colore al nei bit 0-2 della mappa colori;

.4)assegnare un colore ausiliario nei bit 4-7 del registro 15;

.5)costruire la mappa caratteri in RAM con i caratteri che si desidera usare.

Segue un esempio; il programma predispone i seguenti colori:
.sfondo giallo,
.bordo nero,
.ausiliario blu.
Viene preparato un carattere come quello prima descritto, in cui compaiono i 4 possibili colori. Questo carattere viene sistemato nella prima posizione della mappa caratteri (in 6144) occupando i primi 8 byte. Il secondo carattere della mappa viene riempito di zeri per poter pulire la mappa video. Poi vengono trasferiti i caratteri necessari per scrivere la parola MULTICOLOR, prelevandoli dalla ROM. Alla linea 53 viene pulito il video, ma dal momento che non è ancora stato sistemato il registro 36869 si vedono apparire tante A. Poi vengono scritti 88 caratteri multicolor usando le POKE e mettendo nella mappa colori il bit 3 al valore 1 e il colore a 0 (nero). Alla linea 70 vengono analogamente scritti altri 88 caratteri in multicolor con il rosso. Dalla 75 alla 78 vengono scritti i caratteri della parola MULTICOLOR in modo normale, cambiando colore ad ogni carattere e in campo inverso. In questo esempio si ha modo di apprezzare il funzionamento multicolor e il funzionamento normale, visti contemporaneamente. Si può provare a modificare alle linee 60 e 70 i codici colori togliendo il bit 3 e quindi usando rispettivamente 0 e 2; facendo girare nuovamente il programma, nelle prime 8 linee del video si possono osservare i caratteri non più in multicolor, ma in modo normale.

```
1 REM MULTICOLOR
10 MV=7680:MC=38400:L=254
15 IFPEEK(648)=16THENMV=4096:MC=37888:L=206
17 PRINT"[illegible]";
20 POKE36879,112:REM SFONDO GIALLO E BORDO NERO
30 POKE36878,80:REM COLORE AUSILIARIO BLU
40 DATA27,27,27,27,27,27,27,27
42 FORK=0TO7:READA:POKE6144+K,A:NEXTK
43 FORK=0TO7:POKE6144+8+K,0:NEXTK
45 K1=13*8:K2=16:GOSUB200
46 K1=21*8:K2=24:GOSUB200
47 K1=12*8:K2=32:GOSUB200
48 K1=20*8:K2=40:GOSUB200
49 K1=9*8:K2=48:GOSUB200
50 K1=3*8:K2=56:GOSUB200:K1=15*8:K2=64:GOSUB200
51 K1=18*8:K2=72:GOSUB200
53 FORK=0TO505:POKEMV+K,1:NEXT
55 POKE36869,L
60 FORK=0TO87:POKEMV+K,0:POKEMC+K,8:NEXTK
70 FORK=88TO175:POKEMV+K,0:POKEMC+K,10:NEXTK
75 Y=MV+226:POKEY,2:POKEY+1,3:POKEY+2,4:POKEY+3,5:POKEY+4,6:POKEY+5,7
76 POKEY+6,8:POKEY+7,4:POKEY+8,8:POKEY+9,9
77 FORK=0TO5:POKEMC+226+K,K+2:NEXTK
78 FORK=0TO4:POKEMC+231+K,K+2:NEXTK
90 GETA$:IFA$=""THEN90
99 END
200 K1=K1+32768:K2=K2+6144
201 FORK=0TO7:POKEK2+K,PEEK(K1+K):NEXTK:RETURN
```

2.6 ESEMPIO GRAFICO DI UNA FUNZIONE

Segue un programma, da usare senza espansioni di memoria, per tracciare il grafico di una funzione senza asintoti.

```
GRAFICO3

0 POKE36869,242:PRINT"JQQQQQQQQI";:INPUT"F(X) ";F$:PRINT"■CQQQQ2DEFFNA(X)="F$
1 PRINT"RUN2■":POKE198,2:POKE631,13:POKE632,13:END
2 DEFFNA(X)=SIN(X)
3 INPUT"JQQQQQCANCELLO IL GRAFICO ";T$
5 IFT$="SI"THENFL=1
6 POKE55,0:POKE56,20:POKE51,0:POKE52,20
10 PRINT"JQQQQQQDOMINIO:":INPUT"DA";ID:INPUT"■A";FD
12 PRINT"NQQQQ   ATTENDERE PREGO■"
15 AD=FD-ID
20 FORX=0TO175:I=ID+X*AD/175:Y=FNA(I):IFIR>YTHENIR=Y
21 IFFR<YTHENFR=Y
22 NEXT:AR=FR-IR
35 IFFL=1THENFL=0:GOSUB300
40 POKE36869,253:POKE36867,22:POKE36865,60:POKE36879,11
70 FORI=0TO241:POKE38400+I,1:NEXT
80 FORJ=0TO21:FORI=0TO10:POKE7680+22*I+J,I+11*J:NEXT:NEXT
82 KK=ABS(IR/AR*87)
85 FORX=0TO175:I=ID+X*AD/175:Y=87-((FNA(I)*87/AR)+KK):GOSUB200:Y=87-KK:GOSUB200
86 NEXT
90 GETA$:IFA$=""THEN90
100 POKE36869,242:POKE36867,46:POKE36865,38:POKE36879,27:RUN
200 E=7-(X-INT(X/8)*8):P=5120+INT(X/8)*88+Y:POKEP,PEEK(P)OR2↑E:RETURN
300 FORI=5120TO7679:POKEI,0:NEXT:RETURN
```

Osservando il listato si vede alla linea 2 la definizione della funzione sin(x); in realtà all'inizio il programma chiede la funzione da calcolare e disegnare e, prima di iniziare il lavoro, va a modificare se stesso, cioè la linea 2, sostituendo alla funzione che c'è già la nuova. Se dopo l'esecuzione si lista il programma si trova modificata la linea 2. La funzione va scritta rispettando le regole del BASIC per le frasi di calcolo. Viene posta la domanda se si vuole cancellare il grafico; la prima volta si deve rispondere "si" ovviamente. Quando il primo grafico è terminato si deve premere RETURN per continuare e si può introdurre una nuova funzione, senza cancellare il vecchio grafico, e ottenere così due grafici sovrapposti. Il problema è che per ogni grafico viene riposizionato l'asse x ed è difficile che per due funzioni diverse gli assi coincidano. Si noti anche l'artificio alla linea 1 per far partire il programma dopo la definizione della funzione con RUN2, scritto come stringa sul video e mandato in esecuzione con le 3 POKE che scrivono nel buffer della tastiera e forzano il comando con il RETURN.

La funzione viene tracciata con il formato dei caratteri 8x8 e con quadro video ridotto a 11 linee. Se si vuole trasformare il programma per farlo funzionare con il VIC espanso bisogna andare a modificare gli indirizzi relativi alle mappe.

CAPITOLO 3

CHIP 6522 E I/O SUL VIC

3.1 CHIP 6522: PIN-OUT

Tutte le funzioni di I/O, escluse quelle relative al video, che è gestito dal 6561, sono realizzate sul VIC tramite due VIA 6522 (Versatile Interface Adapter).
Questi integrati sono interfacce programmabili particolarmente potenti e flessibili, come dice non a torto il loro nome, e presentano, almeno a prima vista, e per chi non abbbia una competenza specifica, una certa complessità. Nelle pagine seguenti sono illustrate le possibilità che offrono, e come vengono utilizzate sul VIC; si rimanda ai manuali dedicati all'argomento per un approfondimento dei dettagli tecnici.
Nella Figura seguente sono riportati sulla destra il pin-out del 6522, e sulla sinistra uno schema dei collegamenti del 6522 con la CPU e le periferiche.

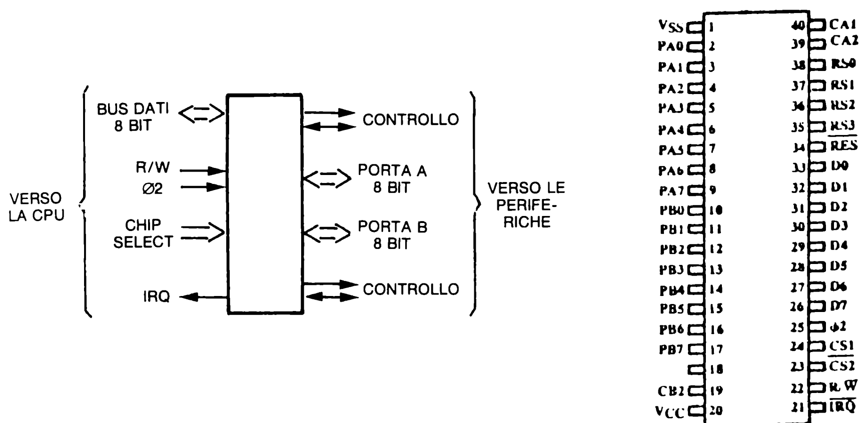


Figura 3.1 Interfaccia 6522 e pin-out

Segue uno schema a blocchi funzionale di un VIA

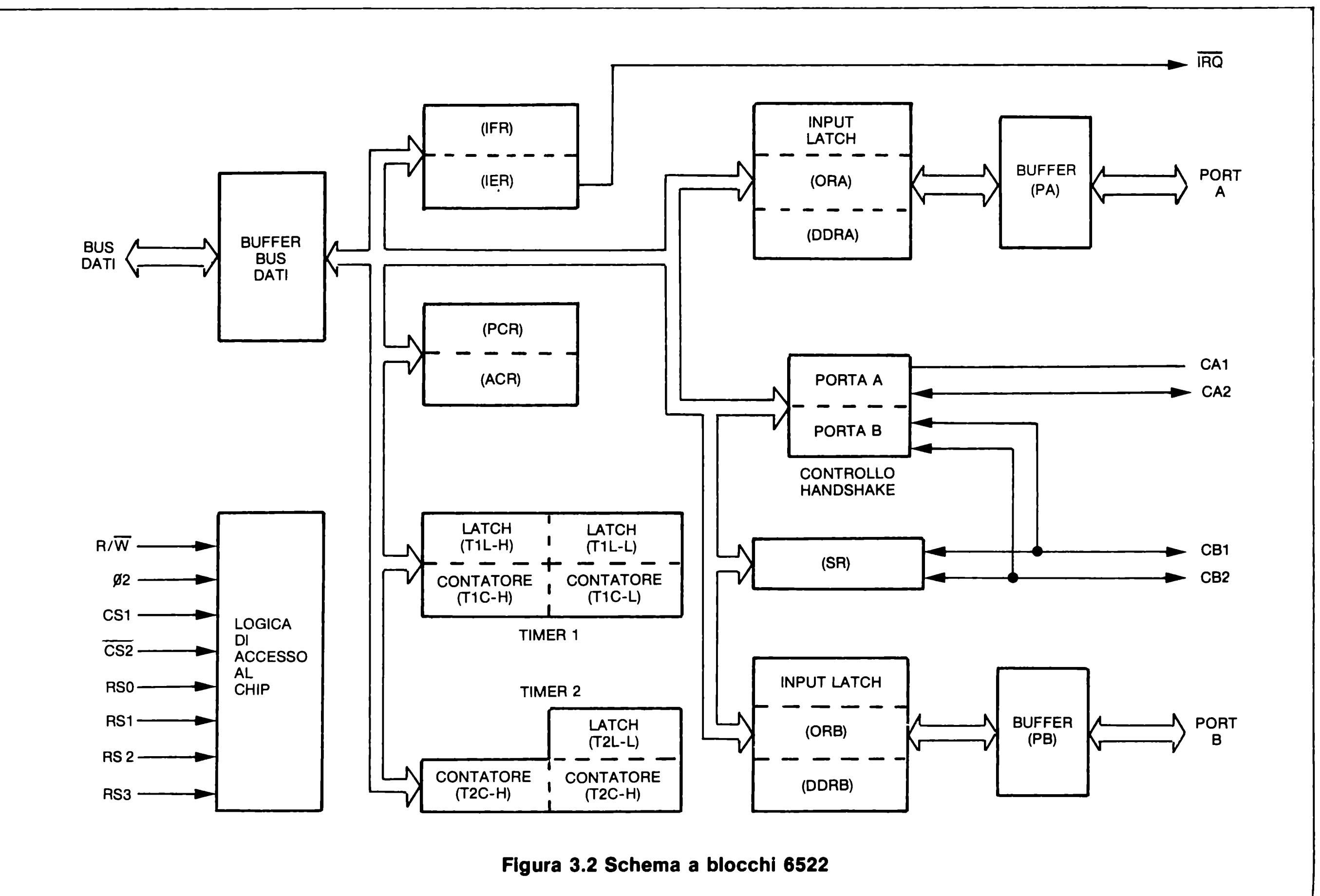


Figura 3.2 Schema a blocchi 6522

Nella sezione di destra sono rappresentate le linee che collegano il VIA alle periferiche (ogni VIA contiene due porte indipendenti per il trasferimento dei dati). Esse sono:
.PA0-PA7: 8 linee bidirezionali, programmabili individualmente sia come ingressi che come uscite, utilizzate per il trasferimento dei dati tra la porta A e le periferiche.
.CA1-CA2: 2 linee di controllo, che possono funzionare sia da linee di interrupt che da segnali di "handshake" (protocollo) nello scambio di dati con una periferica. In questo caso, CA1 funziona da ingresso, e CA2 da uscita.
.PB0-PB7: 8 linee bidirezionali, programmabili individualmente sia come ingressi che come uscite, utilizzate per il trasferimento di dati tra la porta B e le periferiche. Sulla linea PB7 è possibile generare un'onda quadra di frequenza programmabile, mentre la linea PB6 può ricevere in ingresso un'onda quadra, il cui numero di impulsi viene contato da un contatore interno del VIA.
.CB1-CB2: hanno la stessa funzione di CA1 e CA2, e in più vengono utilizzate per l'I/O seriale.

Nella sezione di sinistra sono mostrate le linee di collegamento tra il VIA e la CPU. Esse sono:
.Φ2: è il segnale di clock generato sul piedino 2 della CPU. I trasferimenti di dati tra CPU e VIA hanno luogo quando questo segnale è alto; sul VIC ha una frequenza di 1.1082 Mhz.
.CS1-CS2: linee di chip-select. Il chip funziona solo quando la prima è alta e la seconda è bassa. La CS1 del VIA n.1 è collegata alla linea A4 del bus degli indirizzi, mentre quella del VIA n.2 è collegata alla linea A5; la linea CS2 di entrambi i VIA è collegata all'uscita di un circuito di decodifica delle linee A8-A15, che viene attivata in corrispondenza della configurazione 37120 (9100H). In conclusione, l'indirizzo di partenza dei registri del VIA n.1 è 37136 (9110H), mentre quello del VIA n.2 è 37152 (9120H).

.RS0-RS3: sono le linee di selezione dei registri interni del VIA, accessibili e programmabili tramite un indirizzo. Essi sono:

- i registri di I/O delle due porte (ORA e ORB; IRA e IRB)
- i registri di direzione dei dati delle due porte (DDRA e DDRB)
- il latch a 16 bit e il contatore a 16 bit del timer 1 (T1L-L, T1L-H, T1C-L, T1C-H)
- il latch a 8 bit e il contatore a 16 bit del timer 2 (T2L-L, T2C-L, T2C-H)
- il registro di shift per la trasmissione seriale (SR)
- il registro di controllo delle periferiche (PCR)
- il registro di controllo ausiliario (ACR)
- il registro di flag degli interrupt (IFR)
- il registro di abilitazione degli interrupt (IER)

I registri possono essere utilizzati con diverse modalità a seconda delle configurazioni presenti su queste linee, che sul VIC sono collegate alle linee A0-A3 del bus degli indirizzi; la tabella seguente riporta le 16 possibilità.

TABELLA 3.1					
RS3	RS2	RS1	RS0	REGISTRI	NOTE
0	0	0	0	ORB/IRB	
0	0	0	1	ORA/IRA	con handshake
0	0	1	0	DDRB	
0	0	1	1	DDRA	
0	1	0	0	T1L-L/T1C-L	scrittura latch lettura contatore
0	1	0	1	T1C-H/T1L-H	T1L-L viene copiato in T1C-L
0	1	1	0	T1L-L	
0	1	1	1	T1L-H	
1	0	0	0	T2L-L/T2C-L	scrittura latch lettura contatore
1	0	0	1	T2C-H/T2L-H	T2L-L viene copiato in T2C-L
1	0	1	0	SR	
1	0	1	1	ACR	
1	1	0	0	PCR	
1	1	0	1	IFR	
1	1	1	0	IER	
1	1	1	1	ORA/IRA	senza handshake

.R/W: quando è alta segnala un'operazione di lettura da parte della CPU di un dato del VIA; quando è bassa segnala un'operazione di scrittura di un dato da parte della CPU nel VIA.

.DB0-DB7: sono le 8 linee del bus dati, sulle quali vengono scambiati i dati tra la CPU e il VIA.

.RES: è la linea di reset; quando è attiva (bassa) azzera tutti i registri del VIA, tranne lo shift-register e i registri dei timer. Come conseguenza di ciò il VIA risulta programmato con tutte le linee dati come ingressi, e gli interrupt sono disabilitati. Nel VIC è collegata al circuito di accensione.

.IRQ: viene attivata (messa a zero) ogni volta che nel VIA si verifica una condizione di interrupt, e il relativo flag di interrupt è settato. La linea IRQ del VIA n.1 è collegata alla linea NMI della CPU, e viene attivata quando si preme il tasto RESTORE della tastiera; la linea IRQ del VIA n.2 è collegata alla linea IRQ della CPU, e genera un interrupt 60 volte al secondo per il lancio della routine di scansione degli interrupt, che, tra l'altro, gestisce la tastiera.

3.2 CHIP 6522: REGISTRI

● Registro di I/O della porta B (ORB/IRB)
VIA n.1: 37136-9110H
VIA n.2: 37152-9120H
I bit corrispondenti a linee programmate come uscite determinano il livello logico di queste linee, mentre quelli corrispondenti a linee programmate come ingressi, ne riflettono il contenuto. In un'operazione di uscita, lo stato dei bit relativi a linee di ingresso non ha effetto sulle stesse; in un'operazione di ingresso, i bit relativi a linee di uscita sono nello stato determinato dal registro di output, e non dai pin corrispondenti. Questo fa sì che la CPU ne veda lo stato corretto, anche nel caso che la linea corrispondente non raggiunga per qualche motivo il valore di tensione riconoscibile come 1.

● Registro di direzione dei dati della porta B (DDRB)
VIA n.1: 37138-9112H
VIA n.2: 37154-9122H
Controlla la direzione dei dati sulle 8 linee della porta B: uno 0 significa che la linea corrispondente è un ingresso, un 1 significa che è un'uscita.

● Registro di I/O della porta A (ORA/IRA)
VIA n.1: 37151-911FH
VIA n.2: 37167-912FH
E' in tutto simile a quello della porta B, con l'unica differenza che in un'operazione di ingresso i bit relativi a linee di uscita riflettono lo stato di queste linee, e non quello dei bit del registro di uscita.

● Registro di I/O della porta A con controllo dei segnali di handshake
VIA n.1: 37137-9111H
VIA n.2: 37153-9121H
Ha la stessa funzione del precedente, con in più il controllo delle linee di handshake. Se si fa un'operazione di lettura utilizzando la linea CA1 per il latch dell'ingresso, viene settato il flag di interrupt corrispondente; questo viene resettato quando la CPU legge il dato dal registro.

● Registro di direzione dei dati della porta A (DDRA)
Ha la stessa funzione del registro corrispondente della porta B.
VIA n.1: 37139-9113H
VIA n.2: 37155-9123H

● Timer 1
E' costituito da un latch e un contatore, entrambi a 16 bit, agli indirizzi seguenti:
byte basso del contatore: VIA n.1: 37140-9114H
VIA n.2: 37156-9124H

byte alto del contatore:	VIA n.1: 37141-9115H
	VIA n.2: 37157-9125H
byte basso del latch:	VIA n.1: 37142-9116H
	VIA n.2: 37158-9126H
byte alto del latch:	VIA n.1: 37143-9117H
	VIA n.2: 37159 -9127H

Come si vede dalle operazioni elencate nella Tabella 3.1, non è possibile scrivere direttamente nel byte basso del contatore, che viene caricato con il contenuto del byte basso del latch quando si scrive nel byte alto del latch e del contatore; l'inizio del conteggio parte da questo istante. E' possibile inoltre modificare il contenuto di entrambi i byte del latch senza alterare quello del contatore.
Da quar.do il contenuto del latch viene trasferito nel contatore, questo viene decrementato alla frequenza del clock applicato al VIA; quando arriva a zero, viene settato un flag di interrupt, e, se gli interrupt sono abilitati, la linea IRQ del VIA viene attivata, cioè messa a zero.
A seconda della configurazione di due bit del registro di controllo ausiliario (ACR6 e ACR7) si possono avere diversi modi di funzionamento, che sono sintetizzati nella Tabella 3.2.

TABELLA 3.2

ACR6	ACR7	MODO DI FUNZIONAMENTO
0	0	singolo interrupt, PB7 disabilitato
0	1	sequenza di interrupt, PB7 disabilitato
1	0	singolo interrupt e impulso su PB7
1	1	sequenza di interrupt e di impulsi su PB7

In sostanza, se il bit ACR6 è a zero, viene generato un singolo interrupt per ogni operazione di caricamento del contatore, quando questo raggiunge lo zero (one-shot mode); dopodichè gli interrupt vengono disabilitati, e il contatore ricomincia a decrementarsi ripartendo dal valore massimo (65535). Questo consente alla CPU di leggerlo, per calcolare il tempo passato da quando si è avuto l'interrupt.
Se il bit ACR6 è a 1, viene generata una sequenza di interrupt (free running mode), a partire dall'istante in cui viene caricato il contatore, ogni volta che raggiunge lo zero, ad una frequenza che dipende dal valore caricato nel latch.
Dopo ogni interrupt il contatore riparte dal valore che si trova nel latch. Se questo rimane invariato, la sequenza di interrupt avrà una frequenza costante; modificando il valore del latch durante il count-down, si possono generare sequenze di interrupt a frequenza variabile.
Il bit ACR7 abilita o meno la generazione di impulsi sul piedino PB7 della porta B, in corrispondenza degli interrupt generati verso la CPU; nel primo modo viene generato un singolo impulso, nel secondo una sequenza di impulsi, con frequenza

costante o variabile, come abbiamo visto per gli interrupt. Si noti che questi impulsi sul piedino PB7 vengono generati anche se non è stato programmato in uscita.

● Timer 2

E' costituito da un unico registro a 16 bit, di cui il byte basso ha sia la funzione di latch che di contatore, mentre il byte alto ha solo quella di contatore. Gli indirizzi sul VIC sono:

byte basso latch/contatore:	VIA n.1: 37144-9118H
	VIA n.2: 37160-9128H
byte alto contatore:	VIA n.1: 37145-9119H
	VIA n.2: 37161-9129H

Il byte basso funziona da latch a sola scrittura, mentre funziona da contatore a sola lettura; il byte alto funziona da contatore e può essere sia scritto che letto. Le operazioni possibili sono analoghe a quelle del timer 1, e sono riportate nella Tabella 3.1.

I modi di funzionamento del timer 2 sono condizionati dal bit ACR5 del registro ausiliario di controllo; se questo è a zero, il timer funziona in one-shot mode, in modo del tutto analogo al timer 1, senza però la possibilità di generare un impulso su PB7. Se ACR5 è a 1, funziona da contatore degli impulsi che arrivano sulla linea PB6 (pulse counting mode). Il conteggio avviene nel modo seguente: da quando si carica un certo valore nel byte alto del contatore, questo si decrementa ad ogni impulso negativo proveniente da PB6; quando raggiunge lo zero, viene generato un interrupt, e il contatore riparte a decrementarsi dal valore massimo. Per generare ulteriori interrupt occorre però riscrivere nel contatore.

● Il registro di shift (SR)

VIA n.1: 37146-911AH
VIA n.2: 37162-912AH

Tramite il registro di shift si può realizzare la trasmissione e la ricezione seriale di dati sulla linea CB2, con un clock esterno applicato al piedino CB1, o con un clock generato internamente e inviato in uscita sullo stesso piedino.

Il funzionamento di questo registro è controllato dai bit 2, 3 e 4 del registro di controllo ausiliario; il bit ACR4 ne determina il funzionamento in uscita o in ingresso, gli altri due il modo di funzionamento, secondo quanto è sintetizzato nella tabella 3.3.

.modo 000: in questo modo è possibile sia scrivere che leggere nel registro di shift, ma l'operazione di shift è disabilitata, così come il flag di interrupt relativo al registro.

.modo 001: gli impulsi di clock sono generati internamente, ad una frequenza determinata dal byte basso del timer 2, e inviati sul piedino CB1. L'operazione di shift viene lanciata da una lettura o da una scrittura nello shift register; i dati acquisiti serialmente vengono introdotti nel bit meno significativo. Dopo 8 impulsi di clock viene attivata la linea IRQ.

TABELLA 3.3			
ACR4	ACR3	ACR2	MODO DI FUNZIONAMENTO
0	0	0	disabilitato
0	0	1	input con clock timer 2
0	1	0	input con clock di sistema
0	1	1	input con clock esterno
1	0	0	output in free running mode; clock timer 2
1	0	1	output con clock timer 2
1	1	0	output con clock di sistema
1	1	1	output con clock esterno

.modo 010: gli impulsi di clock hanno una frequenza pari alla metà di quella del clock del VIA; per il resto, il modo di funzionamento è analogo al precedente.

.modo 011: CB1 funziona da ingresso degli impulsi di clock generati da un dispositivo esterno; l'operazione di ingresso seriale avviene a questa frequenza, e ad ogni 8 bit viene attivata la linea IRQ del VIA, pur continuando l'operazione di shift. Un'operazione di lettura o scrittura del registro di shift resetta il flag di interrupt e inizializza nuovamente il registro di shift, per l'acquisizione di altri 8 bit. L'operazione di shift viene fatta sempre dal bit meno significativo a quello più significativo.

.modo 100: il contenuto dello shift register viene "shiftato", a partire dal bit più significativo verso quello meno significativo, sulla linea CB2, con un clock determinato dal contenuto del byte basso del timer 2; inoltre i bit rientrano nel registro con una rotazione, e quindi il contenuto del registro stesso viene emesso sulla linea CB2 in modo ripetitivo.

.modo 101: è analogo al precedente, con la differenza che il contenuto dello shift register viene shiftato una sola volta, dopodichè viene settato il flag di interrupt.

.modo 110: è analogo ai precedenti, con il clock determinato solo dalla frequenza del clock del VIA.

.modo 111: la frequenza dell'operazione di shift è determinata dagli impulsi di clock applicati da un dispositivo esterno sul piedino CB1; per il resto, il modo di funzionamento è analogo ai due precedenti.

● Registro dei flag di interrupt (IFR)
VIA n.1: 37149-911DH
VIA n.2: 37165-912DH
● Registro di abilitazione degli interrupt (IER)
VIA n.1: 37150-911EH
VIA n.2: 37166-912EH
Questi due registri controllano gli interrupt generati dal VIA verso la CPU.

I bit da 0 a 6 del primo registro funzionano da flag di interrupt, ognuno relativo ad un certo evento che si verifica nel VIA. Ogni bit viene settato dal verificarsi dell'evento corrispondente, e resettato da una certa operazione. Quando un flag viene settato, se il bit che occupa la stessa posizione nel secondo registro è a 1, viene attivata la linea IRQ del VIA, e quindi viene generata un'interruzione; se invece il bit del registro di abilitazione è a zero, l'interrupt non viene generato.
Il bit 7 segnala lo stato della linea IRQ del VIA; quando è a 1 significa che è stato generato un interrupt.
Le condizioni che influenzano i bit del IFR sono le seguenti:

bit 0:	1 = transizione attiva su CA2
	0 = lettura o scrittura di ORA/IRA
bit 1:	1 = transizione attiva su CA1
	0 = lettura o scrittura di ORA/IRA
bit 2:	1 = fine di uno shift di 8 bit
	0 = lettura o scrittura di SR
bit 3:	1 = transizione attiva su CB2
	0 = lettura o scrittura di ORB/IRB
bit 4:	1 = transizione attiva su CB1
	0 = lettura o scrittura di ORB/IRB
bit 5:	1 = azzeramento del timer 2
	0 = lettura del byte basso o scrittura del byte alto del timer 2
bit 6:	1 = azzeramento del timer 1
	0 = lettura del byte basso o scrittura del byte alto del timer 1

I flag del IFR possono essere settati solo dal verificarsi della condizione corrispondente, mentre possono essere azzerati, oltre che dalle condizioni già viste, anche direttamente dalla CPU, con un'operazione di scrittura nel registro; in questo caso, un 1 azzera il flag corrispondente, mentre uno zero lo lascia inalterato. Inoltre la CPU, può naturalmente, leggere il contenuto del registro. Il bit 7 di questo registro non può essere azzerato direttamente, ma solo azzerando tutti i flag di interrupt, o quelli di abilitazione degli interrupt.
I flag del secondo registro (IER) possono essere settati o resettati dalla CPU individualmente, inviando un byte al registro stesso. Se il bit 7 di questo byte è a zero, i bit da 6 a 0 posti a 1 azzerano i bit corrispondenti del IER, e quelli a 0 li lasciano inalterati; se il bit 7 è a 1, i bit da 6 a 0 posti a 1 settano il flag di abilitazione corrispondente, mentre quelli a 0 li lasciano inalterati. Il registro IER può inoltre essere letto dalla CPU; in questa operazione il bit 7 risulta sempre a 0.

● Registro di controllo delle periferiche (PCR)
VIA n.1: 37148-9110H
VIA n.2: 37164-9120H
Seleziona il modo di funzionamento delle 4 linee di controllo delle periferiche; le funzioni dei vari bit sono illustrate nella figura seguente.

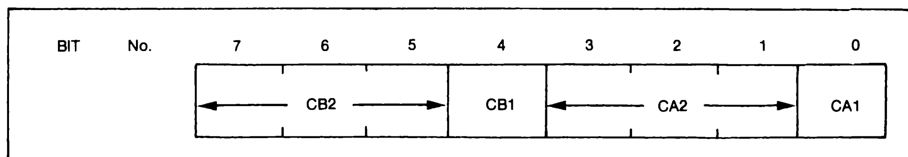


Figura 3.3 Registro di controllo delle periferiche

Il bit di controllo di CA1 e di CB1 definisce la "transizione attiva" (da 0 a 1, o da 1 a 0), cioè quella che setta il corrispondente flag di interrupt.
I bit di controllo di CA2 e CB2 stabiliscono se queste linee funzionano da ingresso di interrupt o da uscita di handshake verso la periferica; nel primo caso selezionano diverse modalità di azzeramneto del flag di interrupt corrispondente, e in entrambi i casi selezionano il senso della transizione attiva.
Si ricorda che CB1 e CB2, nel caso sia abilitata la porta di uscita seriale, funzionano rispettivamente da linee di clock e linea di trasmissione seriale.

● Registro di controllo ausiliario
VIA n.1: 37147-911BH
VIA n.2: 37163-912BH
Le funzioni di controllo dei singoli bit sono riassunte nella figura seguente.

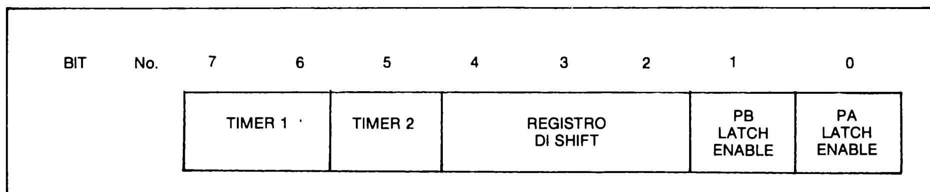


Figura 3.4 Registro di controllo ausiliario

I modi di funzionamento dei registri controllati da questi bit sono stati illustrati in precedenza.

3.3 I/O SUL VIC

Il VIA 6522 può essere utilizzato, come si è visto nel paragrafo precedente, per una gran varietà di operazioni di I/O, e quindi collegato a diversi tipi di periferiche. Il VIC possiede 2 VIA; per ognuno di essi sono stati dati gli indirizzi relatvi ai vari registri. Fermo restando che l'utente è libero di utilizzare questi chip per sue particolari operazioni di I/O, collegando opportunamente le periferiche, e dotando il calcolatore del software necessario per la loro gestione, si vedrà ora come sono

utlizzati questi stessi chip nella configurazione standard del VIC e con il software di cui è dotato normalmente.
Le funzioni realizzate tramite i VIA si possono così riassumere:

- input da tastiera
- I/O su registratore a cassetta
- I/O seriale con interfaccia IEEE 488
- I/O seriale con interfaccia RS232
- input da joystick.

In questo Capitolo si fa anche un breve cenno all'input da paddle e da light pen, in quanto, anche se non sono gestiti dai VIA, rappresentano tipiche operazioni di I/O del VIC.
La figura seguente illustra l'utilizzo delle linee di I/O dei 2 VIA sul VIC.

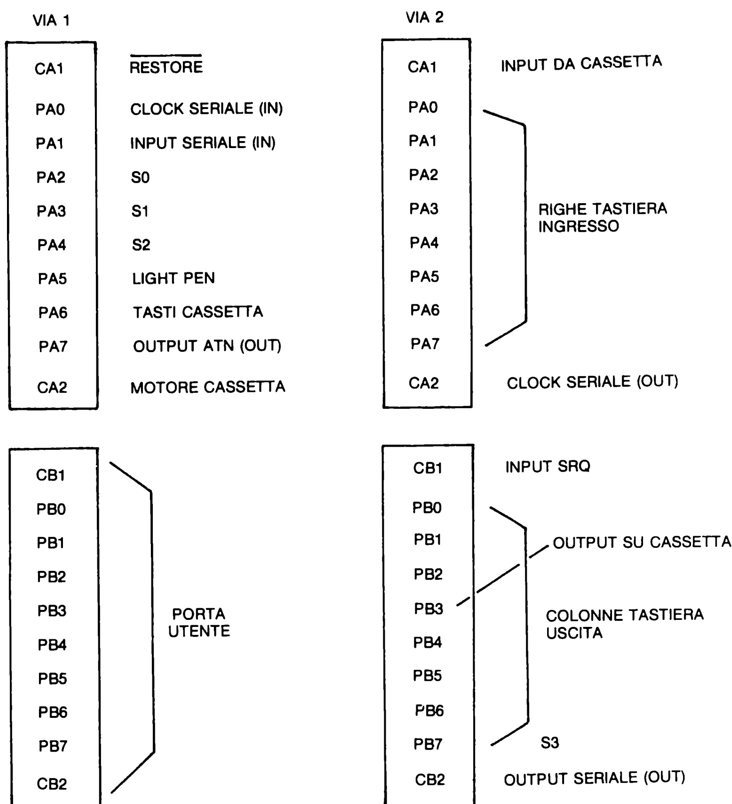


Figura 3.5 Linee di I/O dei due chip 6522

La figura seguente illustra la posizione sul VIC dei diversi connettori per l'I/O.

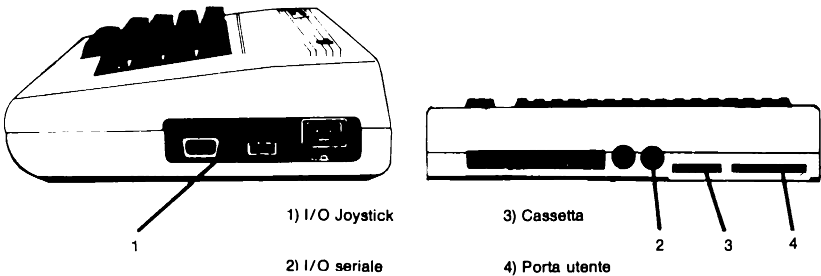
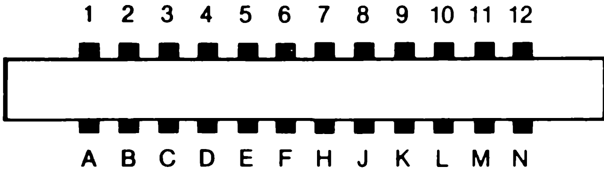


Figura 3.6 Connettori di I/O sul VIC

Il VIA n.2 serve per l'input da tastiera, ed è perciò inaccessibile all'utente; il VIA n.1 è utilizzato per diverse operazione di I/O, ed è acccessibile all'utente tramite un connettore a 24 pin, di cui si riporta di seguito il pin-out.



PIN	FUNZIONE	PIN	FUNZ.
1	GND	A	GND
2	+ 5 V	B	CB1
3	RESET	C	PB0
4	S0	D	PB1
5	S1	E	PB2
6	S2	F	PB3
7	LIGHT PEN	H	PB4
8	TASTI CASSETTA	J	PB5
9	ATN IN	K	PB6
10	+ 9 V	L	PB7
11	GND	M	CB2
12	GND	N	GND

Figura 3.7 Connettore porta utente

3.3.1 INPUT DA TASTIERA

I tasti della tastiera sono 66, di cui 64 rappresentano caratteri alfanumerici o funzioni speciali, gli altri due sono il RESTORE e lo SHIFT LOCK. Quest'ultimo è

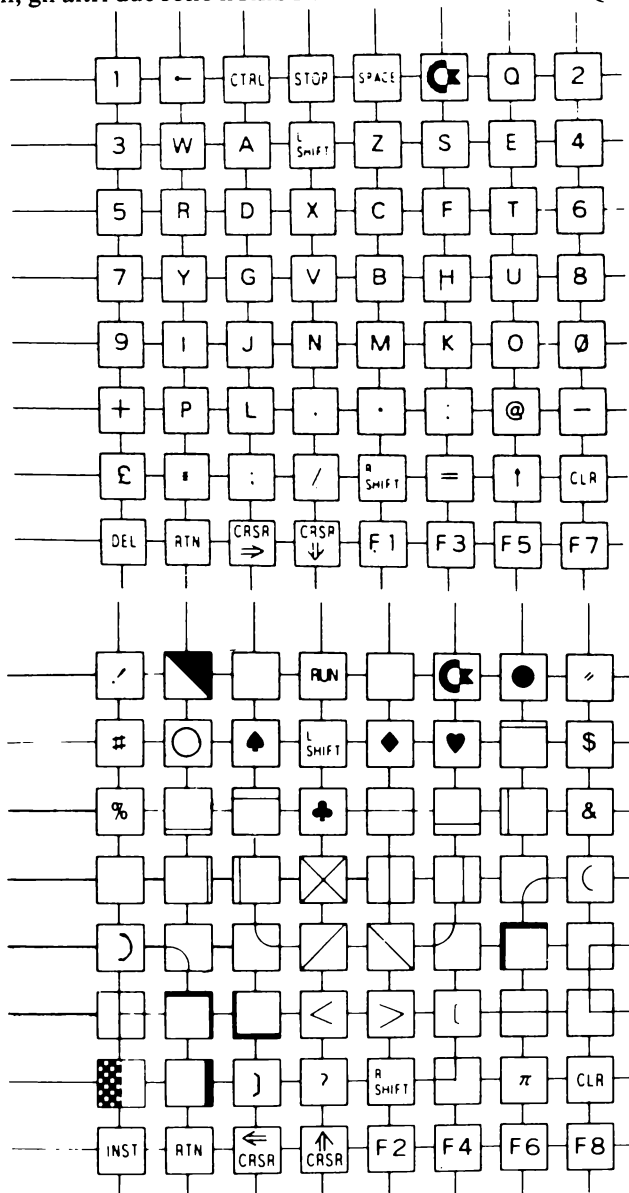


Figura 3.8 Matrici connessioni tastiera

un dispositivo meccanico che serve per tenere premuto stabilmente lo SHIFT; non viene interrogato dal calcolatore, che invece interroga lo SHIFT, nelle normali operazioni di scansione della tastiera.

Il tasto RESTORE è direttamente collegato al piedino CA1 del VIA n.1; la sua pressione mette a zero questa linea, provocando un'interruzione sulla linea NMI della CPU, a cui è collegata la linea IRQ del VIA.

I 64 tasti alfanumerici sono collegati alle 16 linee di I/O della porta A e della porta B del VIA n.2, in modo da costituire gli elementi di una matrice 8x8, in cui le righe sono le linee della porta A, e le colonne quelle della porta B.

La porta A è programmata in ingresso, la porta B in uscita; la scansione della tastiera avviene 60 volte al secondo, in risposta a un'interruzione generata appunto con questa frequenza dal VIA n.2, tramite il timer 1, sulla linea IRQ della CPU. La routine di servizio di questo interrupt inizia alla locazione 60095 (EABFH); la parte di scansione inizia alla locazione 60180 (EB1EH), e funziona sul seguente principio: la pressione di un tasto determina un contatto tra la linea della porta A e quella della porta B al cui incrocio si trova il tasto, e quindi lo stato logico inviato in uscita sulla linea della porta B si ritrova in ingresso sulla linea della porta A.

La routine di scansione comincia con l'accertare se c'è un tasto premuto, e in caso contrario termina. Se almeno un tasto è stato premuto, la matrice viene scandita nuovamente per identificarlo; ad ogni tasto è associato un numero che ne individua la posizione nella matrice, e che fa da puntatore a una delle 4 tabelle dei caratteri della tastiera; la prima di queste si trova alla locazione 60510 (EC5EH). Nelle tabelle sono registrati i valori ASCII corrispondenti ai vari tasti. La corrispondenza si ha per mezzo di un puntatore che può assumere un valore da 0 a 63 se è premuto un tasto, 64 se non è premuto alcun tasto. Le corrispondenze tra i valori del puntatore, i codici ASCII dei tasti e i tasti sono riportati nella Tabella 3.4 per la prima tabella.

Più precisamente, una volta individuato il tasto premuto e calcolato il numero che lo identifica, la routine accerta se oltre a questo è premuto il tasto SHIFT, il tasto CBM o entrambi, e a seconda dei diversi casi, viene modificato l'indirizzo di inizio della tabella dei caratteri. Il codice ASCII del tasto premuto viene poi introdotto nel buffer della tastiera, che si trova dalla locazione 631 alla 640 (0277H-0280H), e viene aggiornato il contatore dei caratteri presenti nel buffer, situato nel byte 198. Nel byte 203 (00CBH) si trova 0 se la tastiera in stato normale, 1 se è premuto SHIFT, 2 se è premuto CBM e 3 se sono premuti insieme SHIFT e CBM. Nel byte 0653 (028DH) si trova il numero corrispondente al tasto premuto (64 se nessun tasto).

Segue il programma CARTAST che lista le 4 tabelle di caratteri. La prima tabella inizia a 60510 (EC5EH); il suo indirizzo può essere ricavato dai 2 byte 245 e 246 della pagina zero. Ogni tabella occupa 65 byte e termina con un byte contenente 255 (FFH), corrispondente al valore 64 del puntatore. Si osservino le tabelle stampate;

TABELLA 3.4								
Punt.	Tasto	ASCII	Punt.	Tasto	ASCII	Punt.	Tasto	ASCII
0	1	49	22	;	59	43	H	72
1	3	51	23	CRSRs/d	29	44	K	75
2	5	53	24	STOP	3	45	:	58
3	7	55	25	niente	1	46	=	61
4	9	57	26	X	88	47	f3	134
5	+	43	27	V	86	48	Q	81
6	lira	92	28	N	78	49	E	69
7	DEL	20	29	,	44	50	T	84
8	freccia	95	30	/	47	51	U	85
9	W	87	31	CRSRs/g	17	52	O	79
10	R	82	32	spazio	32	53	@	64
11	Y	89	33	Z	90	54	su	94
12	I	73	34	C	67	55	f5	135
13	P	80	35	B	66	56	2	50
14	*	42	36	M	77	57	4	52
15	RETURN	13	37	.	46	58	6	54
16	niente	4	38	niente	1	59	8	56
17	A	65	39	f1	133	60	0	48
18	D	68	40	niente	2	61	-	45
19	G	71	41	S	83	62	HOME	19
20	J	74	42	F	70	63	f7	136
21	L	76						

in esse le colonne rappresentano alternativamente il valore del puntatore e il contenuto del byte puntato (non viene stampato l'ultimo byte).

```
1 REM CARTAST
10 OPEN4,4:CMD4
20 GOSUB200
90 PRINT#4:CLOSE4
100 STOP
200 FORI=0TO3
205 X=PEEK(245)+256*PEEK(246)
206 X=X+I*65
207 PRINT"TABELLA N. ";I+1:PRINT
210 FORK=0TO63STEP4
220 FORJ=0TO3
230 PRINTRIGHT$("    "+STR$(K+J),3);"  ";
235 PRINTRIGHT$("    "+STR$(PEEK(K+J+X)),4);"   ";
240 NEXTJ:PRINT:NEXTK
245 PRINT:PRINT:NEXTI
250 RETURN
```

Risultati del programma:

TABELLA N. 1

0	49	1	51	2	53	3	55
4	57	5	43	6	92	7	20
8	95	9	87	10	82	11	89
12	73	13	80	14	42	15	13
16	4	17	65	18	68	19	71
20	74	21	76	22	59	23	29
24	3	25	1	26	88	27	86
28	78	29	44	30	47	31	17
32	32	33	90	34	67	35	66
36	77	37	46	38	1	39	133
40	2	41	83	42	70	43	72
44	75	45	58	46	61	47	134
48	81	49	69	50	84	51	85
52	79	53	64	54	94	55	135
56	50	57	52	58	54	59	56
60	48	61	45	62	19	63	136

TABELLA N. 2

0	33	1	35	2	37	3	39
4	41	5	219	6	169	7	148
8	95	9	215	10	210	11	217
12	201	13	208	14	192	15	141
16	4	17	193	18	196	19	199
20	202	21	204	22	93	23	157
24	131	25	1	26	216	27	214
28	206	29	60	30	63	31	145
32	160	33	218	34	195	35	194
36	205	37	62	38	1	39	137
40	2	41	211	42	198	43	200
44	203	45	91	46	61	47	138
48	209	49	197	50	212	51	213
52	207	53	186	54	222	55	139
56	34	57	36	58	38	59	40
60	48	61	221	62	147	63	140

TABELLA N. 3

0	33	1	35	2	37	3	39
4	41	5	166	6	168	7	148
8	95	9	179	10	178	11	183
12	162	13	175	14	223	15	141
16	4	17	176	18	172	19	165
20	181	21	182	22	93	23	157
24	131	25	1	26	189	27	190
28	170	29	60	30	63	31	145
32	160	33	173	34	188	35	191
36	167	37	62	38	1	39	137
40	2	41	174	42	187	43	180
44	161	45	91	46	61	47	138
48	171	49	177	50	163	51	184
52	185	53	164	54	222	55	139
56	34	57	36	58	38	59	40
60	48	61	220	62	147	63	140

TABELLA N. 4

0	201	1	14	2	208	3	11
4	169	5	2	6	13	7	5
8	144	9	141	10	5	11	144
12	76	13	220	14	230	15	201
16	142	17	208	18	11	19	169
20	253	21	45	22	5	23	144
24	141	25	5	26	144	27	76
28	220	29	230	30	201	31	8
32	208	33	10	34	169	35	128
36	13	37	145	38	2	39	141
40	145	41	2	42	48	43	239
44	201	45	9	46	208	47	235
48	169	49	127	50	45	51	145
52	2	53	141	54	145	55	2
56	16	57	225	58	232	59	181
60	217	61	9	62	128	63	149

Si consideri, come esempio, il valore 17 del puntatore nelle 4 tabelle: nella prima ad esso corrisponde 65, codice ASCII di a, nella seconda 193, codice ASCII del carattere grafico "cuore", nella terza 176, codice ASCII del carattere grafico a sinistra nel tasto di a e nella quarta 208. Chi volesse approfondire questo argomento può stampare la routine di scansione della tastiera con VICMON, che inizia all'indirizzo EB1EH.

Un problema è rappresentato dal fatto che, durante le operazione di I/O seriale, ad esempio con il registratore a cassetta, il vettore di interrupt relativo alla linea IRQ della CPU viene modificato, e quindi le operazioni di scansione della tastiera vengono sospese; l'unico comando che l'utente può usare in questa situazione è il tasto STOP, perchè la rilevazione della pressione di questo tasto è separata dal resto, ed è presente anche nelle routine di I/O da cassetta, e comunque in tutte le routine di servizio della IRQ. La pressione di questo tasto viene dunque rilevata in ogni caso, e ridà il controllo alla tastiera.

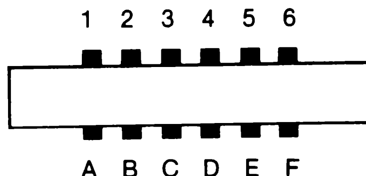
3.3.2 I/O SU REGISTRATORE A CASSETTA

Il registratore a cassetta si collega al VIC tramite i VIA, nel modo seguente:

- VIA n.1, CA2: comando motore
- VIA n.1, PA6: tasti cassetta
- VIA n.2, PB3: scrittura dati
- VIA n.2, CA1: lettura dati

Il connettore per il registratore a cassette, fornisce inoltre due linee con l'alimentazione e la terra.

Si noti che la linea PA6, collegata ai tasti della cassetta, è sensibile alla pressione su uno qualsiasi dei tre (play o avanzamento, rewind o riavvolgimento, fast forward o avanzamento veloce), mentre in realtà dovrebbe rilevare solo la pressione del tasto play, in un'operazione di lettura o scrittura.

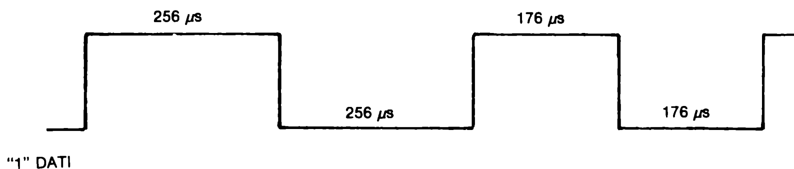


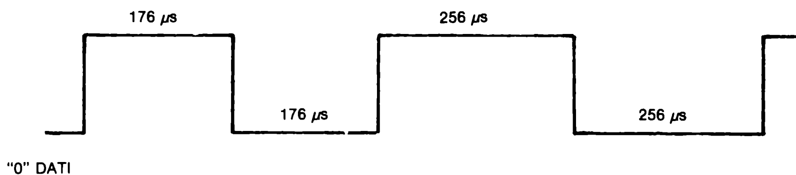
PIN	FUNZIONE
A-1	GND
B-2	+ 5 V
C-3	MOTORE CASSETTA
D-4	LETTURA DATI
E-5	SCRITTURA DATI
F-6	TASTI CASSETTA

Figura 3.9 Pin-out del connettore della cassetta

Le operazioni di lettura e scrittura sono gestite quasi completamente via software dal sistema operativo, che codifica gli 0 a gli 1 usando tre onde quadre di diversa frequenza, e precisamente:

.impulsi di lunga durata, con una frequenza di 1488 Hz
.impulsi di media durata con una frequenza di 1953 Hz
.impulsi di breve durata, con una frequenza di 2840 Hz





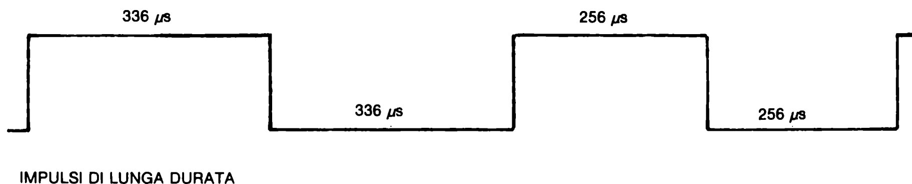
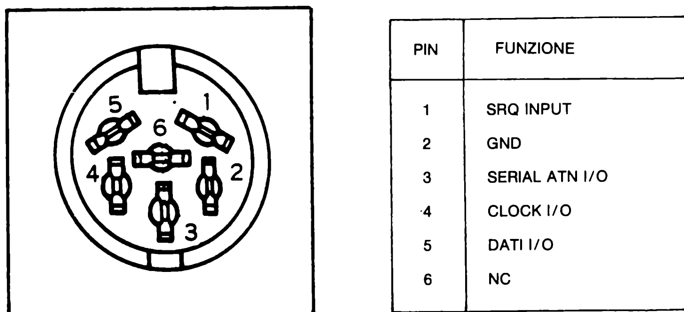


Figura 3.10 Forma d'onda del registratore a cassetta

Un byte di dati coinvolto in un'operazione di registrazione è costituito di 8 bit di dati, un bit di "word marker", e un bit di parità. Gli 8 bit di dati vengono codificati con un impulso medio, seguito da uno breve, se sono a 1, e viceversa se sono a 0; la durata totale della registrazione di un bit di dato è dunque di 864 microsecondi. Il bit di parità è codificato con un impulso lungo seguito da uno breve, o viceversa, a seconda del suo stato, mentre il bit di "word marker" è codificato con sequenze di impulsi lunghi e medi.
I tempi di registrazione rilevati durante un'operazione reale sono notevolmente diversi da quelli che si potrebbero prevedere in base alle informazioni precedenti; questo perchè, come si preciserà nel capitolo sui file su cassetta, i dati sono sempre regitrati due volte, e tra due blocchi successivi di 192 byte viene lasciato un gap di registrazione, il che rallenta le operazioni.
Le sequenze di impulsi con cui vengono registrati e letti i dati sono realizzate tramite i timer 1 e 2 del VIA n.2.

3.3.3 I/O SU DISCO E STAMPANTE: BUS IEEE

Le comunicazioni tra il VIC e le periferiche come la stampante e il disco avvengono serialmente attraverso un bus IEEE collegato ad alcune linee dei VIA, con le seguenti funzioni:
.VIA n.1, PA1: input dati
.VIA n.1, PA0: input clock
.VIA n.1, PA7: ATN (Attention) output
.VIA n.2, CB2: output dati
.VIA n.2, CA2: output clock
.VIA n.2, CB1: SRQ (Service Request) input
Il connettore del bus IEEE si presenta come nella figura seguente.



PIN	FUNZIONE
1	SRQ INPUT
2	GND
3	SERIAL ATN I/O
4	CLOCK I/O
5	DATI I/O
6	NC

Figura 3.11 Pin-out del connettore della porta IEEE

Come si vede dalla tabella, ci sono tre linee per l'output dei dati e tre per l'input; il vantaggio del bus IEEE rispetto ad altri dispositivi di trasmissione seriale è che alle sue linee possono essere collegati diversi dispositivi, che con una semplice tecnica di

"protocollo" possono essere messi in comunicazione, naturalmente due alla volta. Ogni dispositivo collegato al bus può assumere uno dei seguenti ruoli:
.controllore (controller)
.ricevitore (listener)
.trasmettitore (talker)
Il software di gestione del VIC è tale per cui solo la CPU può esercitare la funzione di controllo, oltre alle altre due, mentre il drive del disco può essere sia trasmettitore che ricevitore, e la stampante solo ricevitore.

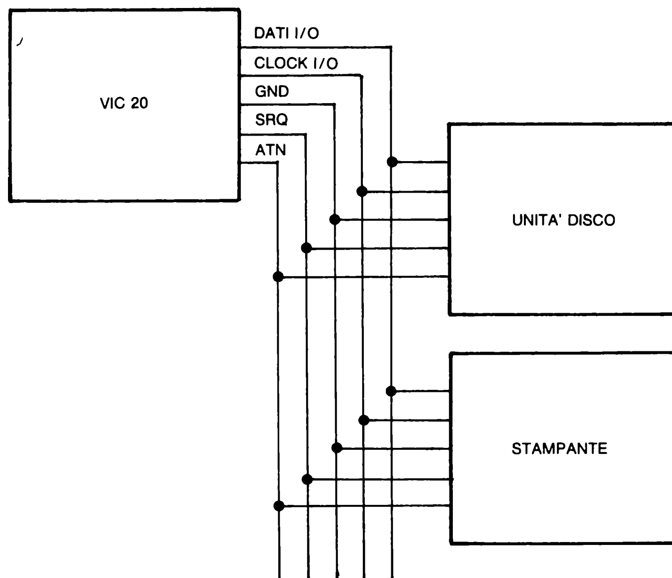
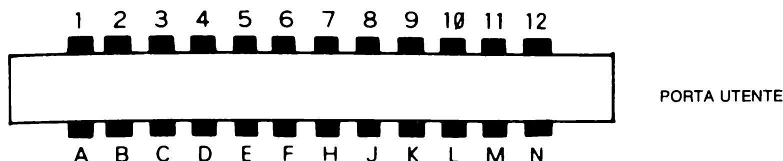


Figura 3.12 Collegamento delle periferiche alla porta IEEE

Inoltre uno dei due dispositivi in comunicazione tramite l'IEEE è sempre la CPU. Il controllore determina di volta in volta qual'è il dispositivo trasmittente, e quale (o quali) quello ricevente, individuando ogni dispositivo con un numero da 4 a 30; dopodichè dà il via alla trasmissione dei dati, che avviene serialmente, un bit dopo l'altro, sulle linee di input o di output dei dati, alla frequenza determinata dai clock di input o di output rispettivamente. Alla fine il bus viene liberato per successive comunicazioni. Questo colloquio avviene con opportuni segnali sulle linee di controllo ATN e SRQ.
I comandi del bus IEEE sono i seguenti:
OPEN INPUT GET PRINT CLOSE LOAD SAVE CMD.
Per una spiegazione più precisa e dettagliata del loro uso si rimanda ai capitoli sul disco e sulla stampante.

3.3.4 RS232

Lo standard RS232 è una linea di trasmissione seriale molto usata nelle comunicazioni tra calcolatore e periferiche; l'I/O per RS232 è generato di solito da una porta di I/O seriale UART(Universal Asynchronous Receiver Transmitter). Sul VIC, invece, questo chip non esiste, e le sue funzioni sono simulate via software su alcune linee del VIA n.1, tramite degli pseudo-registri rappresentati da parole di memoria. Il connettore per RS232 è mostrato nella figura seguente.



PIN	FUNZIONE	RS232
B	CB1	┐
C	PB0	┘ INPUT DATI
D	PB1	— RTS
E	PB2	— DTR
F	PB3	— RI
H	PB4	— CD
K	PB6	— CTS
L	PB7	— DSR
M	CB2	— OUTPUT DATI
N	GND	— GND

Figura 3.13 Pin-out connettore RS-232

La ragione del successo dello standard RS232 sta nel fatto che le informazioni trasmesse sono completamente asincrone, cioè i vari bit che compongono un byte non hanno bisogno di segnali di temporizzazione per essere riconosciuti; questo rende più veloce la trasmissione. Inoltre sono possibili diverse velocità di trasmissione, con diverse opzioni relative all'handshake, al bit di parità, alla lunghezza della parola, ed altro ancora.
Il formato della trasmissione sul VIC è leggermente diverso da quello previsto per lo standard RS232 , ma questo inconveniente può essere semplicemente superato con un circuito di accoppiamento tra le linee del VIC e quelle esterne.
Il formato della trasmissione asincrona mediante RS232, sul VIC, può essere così descritto:
1)nello stato di attesa, quando cioè non si trasmettono dati, ma il collegamento è

effettuato, la linea di uscita viene mantenuta a livello logico alto (nel VIC 5 volt, nello standard 12 volt);
2)una transizione da alto a basso significa che sta arrivando una parola (normalmente 8 bit), e che i successivi livelli di tensione della linea devono essere “letti” dal dispositivo che ascolta;
3)dopo l’ultimo bit trasmesso, la linea torna alta per 1 o 2 cicli; ciò permette al dispositivo che ascolta di prepararsi ad aspettare la prossima transizione da alto a basso, che verrà riconosciuta come avviso del fatto che è in arrivo un’altra parola.
La sintassi con cui si gestisce la trasmissione su RS232 è del tutto simile a quella usata dal VIC per le sue periferiche. Il numero di periferica è 2; la sintassi di apertura è:

```
OPEN lfn,2,0,CHR$(ctrl)+CHR$(cmd)
```

dove

- lfn è il numero di file logico (da 0 a 255); se è maggiore di 127, ogni volta che viene inviato un RETURN, vengono in realtà inviati 2 caratteri: RETURN (CHR$(13)) e LINEFEED (CHR$(10)).
- ctrl è il numero contenuto nel registro di controllo
- cmd è il numero contenuto nel registro di comando

Il registro di controllo è simulato sul VIC da una parola di memoria che si trova all’indirizzo 659-0293H, i cui bit hanno il seguente significato:

bit 7: numero dei bit di stop: 0=1; 1=2

bit 6-5: lunghezza della parola:
00=8 bit; 01=7 bit; 10=6 bit; 11=5 bit

bit 4: non usato

bit 3, 2, 1, 0: velocità di trasmissione

0001 =	50	baud (bit/sec)
0010 =	75	baud
0011 =	110	baud
0100 =	134.5	baud
0101 =	150	baud
0110 =	300	baud
0111 =	600	baud
1000 =	1200	baud
1001 =	2400	baud
1010 =	2400	baud

Il registro di comando è simulato sul VIC da una parola di memoria che si trova

all'indirizzo 660-0294H, i cui bit hanno il seguente significato:

bit 7,6,5:	parità XX0 = parità disabilitata 001 = parità pari 011 = parità dispari 101 = bit di parità 1 111 = bit di parità 0
bit 4:	duplex 0 = full duplex 1 = half duplex
bit 3, 2, 1:	non usati
bit 0:	handshake 0 = tre linee 1 = x linee

Si riassume qui di seguito brevemente il significato di queste opzioni:

● full duplex: significa che i due dispositivi possono simultaneamente ricevere e trasmettere dati su RS232;

● half duplex: significa che le informazioni possono essere trasmesse solo alternativamente dall'uno o dall'altro dispositivo;

● handshake: tale modo permette ai dispositivi di disporre di segnali di controllo, come:
"RTS" (Request To Send, uscita): avvisa il dispositivo ricevente che chi trasmette ha pronto un dato;
"CTS" (Clear To Send, ingresso): indica che l'ascoltatore è pronto a ricevere il dato; viene attivato dall'ascoltatore dopo aver ricevuto un RTS;
"DSR" (Data Set Ready, ingresso): significa che il ricevente è pronto a ricevere dati;
"DTR" (Data Terminal Ready, uscita): è analogo al DSR, e significa "terminale pronto";
"CD" (Carrier Detect, ingresso): è usato solo per il collegamento via modem, il quale pone alta questa linea quando riconosce la frequenza portante trasmessa dall'altro modem;
"RI" (Ring Indicator, ingresso): va collegato al campanello del telefono per effettuare automaticamente un collegamento via modem; la linea deve però essere gestita dall'utente.

Quando viene aperto un canale RS232, il VIC gli riserva un buffer di 512 byte nella posizione più alta della memoria utente; il comando OPEN dovrà quindi essere dato prima di un qualunque comando di assegnamento di variabili, pena la perdita delle variabili stesse (infatti l'apertura del canale produce un CLR). Inoltre non si può aprire più di un canale per RS232 in quanto l'apertura del secondo cancellerebbe il buffer riservato al primo.
La sintassi e i risultati degli altri comandi di gestione file (PRINT#, INPUT#, GET#, CMD) rimangono invariati.

Per collegare due VIC tramite RS232 occorre:
1) collegare il contatto A del connettore per RS232 alla calza metallica di un cavetto schermato con due conduttori in entrambi i computer;
2) collegare il primo conduttore con i contatti B e C da un lato, col contatto M dall'altro;
3) eseguire l'operazione simmmetrica con il secondo conduttore.

Il programma che segue permette di inviare messaggi da un VIC all'altro; in un successivo paragrafo è descritto un programma, leggermente più complicato, che permette di giocare alla battaglia navale tra due VIC collegati in questo modo.

```
1 REM TERMINALE
10 OPEN2,2,0,CHR$(138)+CHR$(32)
20 GET#2,A$:PRINTA$;
30 GETA$:PRINTA$;:IFA$<>""THENS$=S$+A$
40 IFA$=CHR$(13)THENPRINT#2,S$;:S$=""
50 GOTO20
```

Nel programma le costanti 138 e 32 della linea 10 hanno il seguente significato:

138 corrisponde in binario a: 1 0 0 0 1 0 1 0
dove bit 7 = 1 significa 2 bit di stop
bit 6-5 = 00 significa 8 bit per parola
bit 4 non conta
bit 3-0 = 1010 significa 2400 baud

32 corrisponde in binario a: 0 0 0 1 0 0 0 0
dove bit 7-5 = 000 significa niente parità
bit 4 = 1 significa half duplex
bit 0 = 0 significa 3 linee.

3.3.5 JOYSTICK E LIGHT-PEN

Un joystick è una leva (o manopola) che, con il suo movimento, può far spostare un oggetto sul video. Esistono due tipi di joystick: quelli a interruttori, che sono i joystick propriamente detti, con i quali è possibile indicare 9 diverse direzioni di spostamento, e quelli a potenziometro (paddle), che permettono, invece, una regolazione più fine dello spostamento, individuando esattamente la posizione dell'oggetto.
I primi, nel VIC, si possono collegare ad alcune linee di un VIA, mentre gli altri sono gestiti dal chip 6561; per comodità verranno trattati entrambi in questo paragrafo.
Gli interruttori dei joystick possono essere rappresentati come nella Figura seguente.

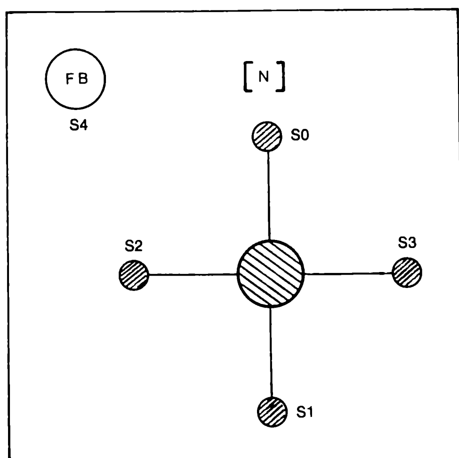


Figura 3.14 Interruttori dei joystick

In base al loro stato, è possibile identificare 9 diverse posizioni della leva:

tutti aperti	leva verticale
S0 chiuso	nord
S1 chiuso	sud
S2 chiuso	ovest
S3 chiuso	est
S0, S1 chiusi	nord-ovest
S1, S2 chiusi	sud-ovest
S2, S3 chiusi	sud-est
S3, S0 chiusi	nord-est

Gli interruttori sono collegati ad alcune linee del VIA, accessibile tramite un connettore, come illustrato nella figura seguente.

PIN	FUNZIONE
1	S0
2	S1
3	S2
4	S3
* 5	POT Y
* 6	LIGHT PEN
7	+ 5 V
8	GND
* 9	POT X

Figura 3.15 Pin-out del connettore del joystick

Per poter leggere con un programma BASIC la posizione della leva, occorre prima di tutto programmare in input le linee del VIA che interessano, e poi leggere il contenuto dei registri di I/O delle porte.
Si riportano di seguito, per comodità, gli indirizzi da usare in queste operazioni.
VIA n.1, DDRA: 37139-9113H
VIA n.1, IRA : 37137-9111H
VIA n.2, DDRB: 37154-9122H
VIA n.2, IRB : 37152-9120H

Il programma seguente inizializza i VIA, e legge nelle variabili S0, S1, S2 e S3 lo stato dei quattro interruttori; la variabile S contiene, invece, lo stato del fire button, che è un pulsante posto in cima alla leva, che può essere premuto per segnalare che la posizione desiderata è stata raggiunta.

```
500 REM J1
501 POKE37139,127:POKE37154,127
502 S=PEEK(37137)
503 S0=((SAND4)=0)
504 S1=((SAND8)=0)
505 S2=((SAND16)=0)
506 FB=((SAND32)=0)
507 S=PEEK(37152)
508 S3=((SAND128)=0)
509 POKE37154,255:RETURN
```

Quest'altro segmento di programma converte le variabili S0, S1, S2 e S3 in valori diversi, corrispondenti alle 9 diverse posizioni della leva, secondo la figura seguente.

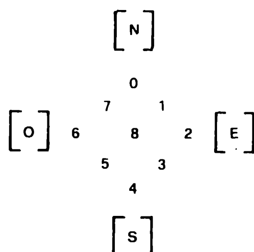


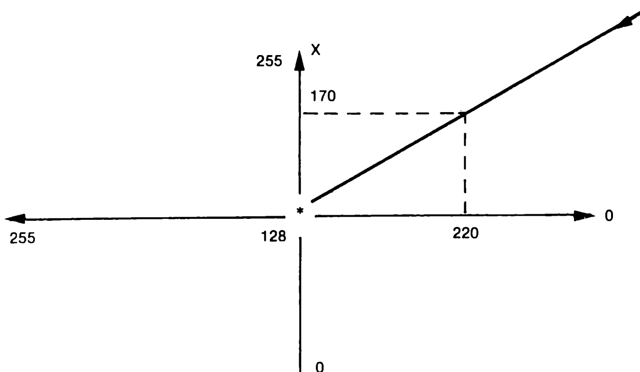
Figura 3.16 Valori attribuiti alle posizioni del joystick

```
1 REM J2
10 DIMJS(2,2)
20 J$="701682543"
30 FORI=0TO2:FORJ=0TO2
40 JS(J,I)=VAL(MID(J$,3*I+J+1,1))
50 NEXTJ,I
500 REM ROUTINE J1
501 POKE37139,127:POKE37154,127
502 S=PEEK(37137)
503 S0=((SAND4)=0)
504 S1=((SAND8)=0)
505 S2=((SAND16)=0)
506 FB=((SAND32)=0)
507 S=PEEK(37152)
508 S3=((SAND128)=0)
509 POKE37154,255
550 X=1+S2-S3:Y=1+S0-S1
560 P=JS(X,Y)
570 RETURN
```

Nei paddle la leva è collegata a due circuiti del tipo schematizzato nella Figura 3.17-a.

Lo spostamento della leva, producendo uno spostamento dei cursori dei due potenziometri, provoca una variazione del valore della resistenza inserita nel circuito, che opportunamente tradotto da analogico a digitale, fornisce sotto forma di un numero a 8 bit, le coordinate x e y della posizione della leva in un ideale piano cartesiano, come mostrato in Figura 3.17-b.

I valori (128,128) corrispondono alla posizione verticale; in totale sono possibili 256 posizioni diverse sull'asse x, e altrettante sull'asse y.



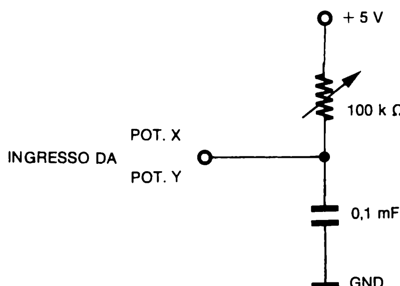


Figura 3.17 Potenziometri

L'uscita dei due potenziometri è collegata a due ingressi del chip 6561, che esegue la conversione e memorizza i due valori nei registri
36872-9008H: potenziometro asse x
36873-9009H: potenziometro asse y

```
1000 REM J3
1010 POKE37139,127:POKE37154,127
1020 PX=PEEK(36872)
1030 FX=-((PEEK(37137)AND16)=0
1040 FY=-((PEEK(37152)AND128)=0
1050 POKE37154,255
1060 PY=PEEK(36873)
1070 RETURN
```

Il programma J3 è una routine per leggere lo stato dei paddle. FX e FY contengono 1 se il pulsante relativo è stato premuto e 0 se non lo è stato. PX e PY danno la situazione delle manopole.

Due parole, infine, sulla light-pen; è un dispositivo sensibile alla luce, la cui posizione sul video può essere identificata dal calcolatore nel modo seguente.

L'immagine sul video viene mantenuta con una continua scansione da parte di un raggio di luce delle righe dello schermo, da sinistra verso destra, una dopo l'altra; quando il fascio di luce passa sotto la light-pen, viene inviato un segnale al chip 6561, il quale memorizza istantaneamente, in due registri, l'attuale posizione verticale e orizzontale del fascio di luce, e quindi della penna luminosa. In questo modo la penna può essere utilizzata per segnalare al calcolatore una posizione sul video.

3.4. ESEMPIO BATTAGLIA NAVALE TRA 2 VIC

Il programma che segue deve essere caricato su entrambi i VIC dopo che sono stati collegati come spiegato precedentemente. E' necessario usare calcolatori con espansione di memoria. Sui due video compaiono i quadri della battaglia, 2 per ogni calcolatore, e rappresentano il quadro proprio e quello dell'avversario. Le coordinate di riferimento sono: le cifre da 0 a 9 per le colonne e le lettere da A a J per le righe.

```
1 REM BATTAGLIA NAVALE
10 OPEN2,2,0,CHR$(130)+CHR$(32)
15 REM 2400 BAUD, 2 STOP BITS, PAROLE 8 BIT; NO PARITA', HALF DUPLEX, TRE LINEE
20 PRINT#2,"0";
25 PRINT"■■■■■■■■■BATTAGLIA NAVALE"
26 PRINT"■■■■■■■■■■IL TUO AVVERSARIO    ■ANCORA NON E' PRONTO"
30 GET#2,Z$:IFZ$=""THEN30:REM ATTENDE UN CARATTERE DALL' ALTRO COMPUTER
33 REM PER SINCRONIZZARSI
40 PRINT#2,"1";:FORI=1TO100:NEXT:GET#2,B$:REM B$ E' LETTA PER SVUOTARE BUFFER
50 POKE36879,57:VM=PEEK(648)*256:X=1:Y=3:REM POSIZIONI INIZIALI CURSORE
58 REM I CARATTERI GRAFICI CHE SEGUONO SONO: <SHIFT>+O E <CBM>+G
60 PRINT"■■■ 0123456789 0123456789A┬┬┬┬┬┬┬┬┬┐ ┌┬┬┬┬┬┬┬┬┬B┬┬┬┬┬┬┬┬┬┐ ┌┬┬┬┬┬┬┬┬┬"
;
61 PRINT"C┬┬┬┬┬┬┬┬┬┐ ";
70 PRINT"┌┬┬┬┬┬┬┬┬┬D┬┬┬┬┬┬┬┬┬┐ ┌┬┬┬┬┬┬┬┬┬E┬┬┬┬┬┬┬┬┬┐ ┌┬┬┬┬┬┬┬┬┬F┬┬┬┬┬┬┬┬┬┐ ┌┬┬┬┬
┌┬┬┬┬";
80 PRINT"G┬┬┬┬┬┬┬┬┬┐ ┌┬┬┬┬┬┬┬┬┬H┬┬┬┬┬┬┬┬┬┐ ┌┬┬┬┬┬┬┬┬┬I┬┬┬┬┬┬┬┬┬┐ ┌┬┬┬┬┬┬┬┬┬";
90 PRINT"J┬┬┬┬┬┬┬┬┬┐ ┌┬┬┬┬┬┬┬┬┬ ────────── ──────────"
91 REM GLI ULTIMI CARATTERI GRAFICI SONO <CBM>+T
100 XY=22*Y+X+VM:BF=PEEK(XY):POKEXY,102:REM BF= CARATTERE SOTTO IL CURSORE
102 REM XY= INDIRIZZO RAM VIDEO OVE SI TROVA IL CURSORE
103 REM DOPO IL POSIZIONAMENTO DELLE NAVI PREMERE RETURN
104 REM PER METTERE E TOGLIERE NAVI PREMERE <SPAZIO>
105 GETA$:IFA$<>"■"ANDA$<>"■"ANDA$<>"■"ANDA$<>"■"ANDA$<>" "ANDA$<>CHR$(13)THEN10
5
106 REM CARATTERI ACCETTATI DURANTE IL POSIZIONAMENTO DELLE NAVI
107 IFA$=CHR$(13)THENPOKEXY,BF:GOTO310
110 IFA$="■"THENY=Y-1:IFY<3THENY=3:REM CURSORE SU
120 IFA$="■"THENY=Y+1:IFY>12THENY=12:REM CURSORE GIU'
130 IFA$="■"THENX=X-1:IFX<1THENX=1:REM CURSORE SINISTRA
140 IFA$="■"THENX=X+1:IFX>10THENX=10:REM CURSORE DESTRA
150 IFA$=" "THEND=-160*(BF<>160)-79*(BF=160):POKEXY,D:BF=D:GOTO105
151 REM PONE NAVE AL POSTO DEL CURSORE, O LA TOGLIE
160 POKEXY,BF:GOTO100:REM PONE AL POSTO DEL CURSORE IL VECCHIO CARATTERE
309 REM INIZIO GIOCO
310 IFZ$="1"THEN400:REM Z$=1 SOLO SE SI E' PREMUTO RUN PER ULTIMI
320 GOSUB10000:PRINT" OK TOCCA A TE!        ■BUONA FORTUNA !!!"
```

```
330 PRINT" COORDINATE ";
332 GETA$:IFA$<"A"ORA$>"J"THEN332:REM ATTENDE COORDINATA VERTICALE
335 PRINTA$" -";
337 GETB$:IFB$<"0"ORB$>"9"THEN337:REM ATTENDE COORDINATA ORIZZONTALE
338 A%=VAL(B$):PRINTA%
350 PS=(VM+78+A%+22*(ASC(A$)-65)):REM PS=POSIZIONE OVE SI E' SPARATO
355 IFPEEK(PS)<>79THENGOSUB10000:PRINT"GIA' FATTO":FORI=1TO1500:NEXT:GOTO320
360 POKE36878,15:POKE36877,250:FORI=45TO8STEP-1:POKE36878,I/3:NEXT:POKE36878,0
361 POKE36877,0:REM RUMORI
370 POKE36878,15:FORI=228TO128STEP-1:POKE36876,I:POKE36878,(I-128)/8+1:FORJ=1TO2
0:NEXT
371 NEXT:POKE36876,0
375 PRINT#2,A$;CHR$(13);A%:REM COMUNICA ALL'ALTRO COMPUTER LE COORDINATE
379 REM ATTENDE RISPOSTA
380 GOSUB10000:INPUT#2,R$:PRINTR$:IF R$="HAI VINTO !!!!!" THENCLOSE2:END
390 C=R$<>"BUCO !!!":POKEVM+78+A%+22*(ASC(A$)-65),160*-C+102*-(NOTC):REM AGGIORN
A MAPPA
400 INPUT#2,A$,A%:REM ATTENDE COORDINATE AVVERSARIO
402 GOSUB10000:PRINTA$" -"A%
405 PS=(VM+67+A%+22*(ASC(A$)-65)):REM PS=INDIRIZZO CELLA IN CUI HA SPARATO L'AVV
ERSARIO
410 IFPEEK(PS)=160THENPOKEPS,79:GOSUB20000:GOTO430:REM COLPITO
420 R$="BUCO !!!":POKEPS,102:POKE36877,250:REM MANCATO
424 REM ORA RUMORI MANCATO
425 FORJ=10TO0STEP-1:POKE36878,J:FORK=1TO150
428 NEXT:NEXT:POKE36877,0
430 PRINT#2,R$:REM COMUNICA RISPOSTA A COLPO APPENA RICEVUTO
440 IFR$<>"HAI VINTO !!!"THEN320
450 GOSUB10000:PRINT"HAI PERSO":CLOSE2:END
9999 REM SUBROUTINE CHE CANCELLA SCHERMO DEDICATO A SCRITTE
10000 PRINT"[illegible]"
10010 PRINT"                        [illegible]                         [illegible]
[illegible]"
10012 RETURN
19999 REM SUBROUTINE CHE GESTISCE RISPOSTA QUANDO UNA NAVE E' COLPITA
20000 R$="COLPITO !!!":QUI QUANDO NAVE COLPITA
20010 POKE36878,15:POKE36877,200:FORI=15TO0STEP-.05
20020 POKE36878,I:NEXT:POKE36877,0
20030 FL=-1:FORI=0TO9:FORJ=0TO9:IFPEEK(VM+67+I*22+J)=160THENFL=0
20040 NEXT:NEXT:IF FL THENR$="HAI VINTO !!!"
20043 REM SE NON CI SONO PIU' NAVI FINISCE IL GIOCO
20050 RETURN
```

Il programma è abbondantemente commentato con REM; se si desidera adoperare VIC senza espansione di memoria, si possono cancellare tutti i commenti accorciando il programma.

CAPITOLO 4

STAMPANTE VIC 1515

4.1 INTRODUZIONE

La stampante VIC 1515 è una stampante grafica in grado di stampare tutti i caratteri del VIC , sia quelli del set grafico che quelli del set maiuscole/minuscole (si veda Appendice H); inoltre dà la possibilità di definire caratteri particolari, costruiti per punti, per produrre disegni e grafici.
Le stesse prestazioni si ottengono collegando al VIC la stampante grafica GP 100 VC SEIKOSHA.
La stampante è dotata al suo interno di un microprocessore che le consente una notevole flessibilità e una vasta gamma di operazioni particolari. La posizione di inizio stampa può essere scelta dall'utente sia a livello di carattere (80 per riga), che a livello di punti (480 per riga); i caratteri possono essere stampati in modo normale o in campo inverso, con ampiezza semplice o doppia, possono essere ripetute colonne di punti. Queste prestazioni si ottengono sfruttando i modi di stampa, descritti nel paragrafo 4.3. La stampante contiene un buffer di stampa di 90 caratteri, il cui riempimento e svuotamento vengono gestiti in modo automatico dal microprocessore interno.
Essa si collega, tramite il bus seriale IEEE 488 e il chip 6522, al calcolatore, con il connettore circolare per l'I/O seriale che si trova sul retro del VIC; se è presente una unità a dischi, la stampante si collega in cascata a questa.
Si raccomanda di eseguire sempre i collegamenti a dispositivi spenti.
Per una descrizione dettagliata della installazione e della messa in opera della stampante, si rimanda al manuale per l'utente che viene fornito insieme ad essa, ricco di illustrazioni esplicative. Si ritiene però utile riassumere qui brevemente le istruzioni più importanti.
Per mettere la stampante in condizioni di funzionare occorre:
.Installare la cartuccia con il nastro; le istruzioni per questa operazione sono schematicamente illustrate sul retro del coperchio marrone anteriore.
.Inserire il foglio di carta; è molto semplice. Il meccanismo è del tutto analogo a quello di una macchina da scrivere; di diverso c'è solo il fissaggio della carta sulle puntine che si trovano ai lati del carrello, sul fronte della stampante (trascinamoduli). Per far avanzare manualmente la carta, basta ruotare la rotellina di destra verso il retro. Si possono spostare i trascinamodulo spingendoli delicatamente verso destra o sinistra per adattarli alla larghezza della carta.

Ora, prima di accendere il calcolatore, si accende la stampante; la testina effettuerà uno spostamento fino a metà del carrello e tornerà in posizione di partenza. Se ciò non succede, si deve spegnere la stampante, controllare che i collegamenti siano corretti, riaccenderla e riprovare. Se proprio non funziona, occorre rivolgersi al rivenditore.

Cosa non bisogna mai fare:
.stampare senza foglio di carta;
.tentare di far tornare indietro la carta, forzando la rotellina di avanzamento;
.collocare la stampante in un ambiente umido o alla luce diretta del sole; ma questo vale anche per il calcolatore e le altre periferiche;
.tenere la stampante senza coperchio per più del necessario; la polvere può recare danni;
.effettuare i collegamenti con la stampante o il calcolatore accesi;
.tentare di spostare manualmente la testina di scrittura.

Se si usa un pacco di fogli continui, ripiegati a soffietto, esso deve essere sistemato in modo da essere allineato con la stampante, altrimenti, a lungo andare, si può inceppare il meccanismo di trascinamento.
Si può effettuare un test sul funzionamento della stampante posizionando l'interruttore che si trova sul retro a sinistra. Esso ha 3 posizioni: 5, 4 e T. Le prime due servono per assegnare il numero di dispositivo alla stampante, che può essere 4 o 5. La posizione T provoca la stampa di tutti i caratteri del VIC ripetitivamente; per fermarla si deve spegnere la stampante o modificare la posizione dell'interruttore. Si può così controllare la bontà della stampa. Eventuali difetti possono dipendere dalle seguenti cause:
.nastro consumato;
.pressione di stampa non ben regolata; può essere aggiustata spostando una levetta che si trova sulla testina di scrittura.
In caso di difetti persistenti si deve contattare il rivenditore.

Le caratteristiche tecniche della stampante sono:
.stampa a matrice di punti 5x7 (su 6x7)
.codici caratteri a 8 bit (del VIC 20)
.dimensioni di un carattere:altezza 2.82 mm,
larghezza 1.76 mm
.velocità: 30 car/sec
5 line feed/sec in modo carattere
7.5 line feed/sec in modo grafico
.80 colonne (480 punti) per riga
.densità di stampa: 12 car/pollice
6 linee/pollice in modo carattere
9 linee/pollice in modo grafico
.possibilità di copie multiple (2 più l'originale).

4.2 COMANDI DI STAMPA

I comandi BASIC necessari per la stampa sono:

OPEN lfn,dn,sa
dove:

.lfn (logical file number) è il numero logico del file che si intende aprire; può essere un qualsiasi intero tra 1 e 255, e deve essere lo stesso numero che compare nei comandi successivi, fino alla chiusura del file;
.dn (device number) è un numero che individua la periferica, e può essere 4 o 5, a seconda della posizione dell'apposito interruttore;
.sa (secondary address) è un numero che stabilisce il modo di operazione della periferica; può assumere i valori:
.0-modo CURSOR UP (set grafico)
.7-modo CURSOR DOWN (set maiuscole/minuscole).

PRINT #lfn,dati
dove:

.lfn è il numero logico del file usato nella OPEN;
."dati" rappresenta l'insieme dei dati e dei caratteri di controllo da stampare.
Dopo la PRINT #, la linea di comunicazione verso la stampante è chiusa, mentre il collegamento come file logico resta aperto. Non si può usare il "?" per la parola chiave PRINT.

CMD lfn,dati
dove:

.vale quanto detto sopra per i parametri.

Con questo comando il controllo di uscita sul video viene trasferito alla stampante, inibendo la normale uscita sul video. Dopo l'esecuzione del comando, la linea di comunicazione verso la stampante resta aperta cioè la stampante permane in stato di attesa dati. Per chiudere la linea si deve usare un comando PRINT # prima della CLOSE.

CLOSE lfn
dove:
.lfn è il numero logico del file.

Chiude il file logico aperto con la OPEN, può essere usato dopo un comando PRINT #, e non dopo un CMD.

Sono riportati di seguito alcuni esempi di utilizzo dei comandi precedenti in modo immediato.

```
OPEN10,4
PRINT#10,"COMMODORE"
CLOSE10

COMMODORE
```

```
OPEN10,4
CMD10
PRINT"COMMODORE"
PRINT#10
CLOSE10

COMMODORE
```

```
OPEN10,4
CMD 10,"COMMODORE"
PRINT#10
CLOSE10

COMMODORE
```

4.3 MODI DI STAMPA

E' possibile produrre una stampa o un listato sia in modo immediato che da programma. L'esempio seguente mostra una sequenza di comandi che producono il listato del programma SP20 prima introdotto. Alla fine del precedente paragrafo si era visto come stampare dati in modo immediato.
Scrivendo quanto segue:

```
OPEN10,4
CMD10
LIST
```

si ottiene il seguente listato (precedentemente digitato):

```
10 OPEN10,4
20 CMD10,"COMMODORE"
```

Per chiudere la linea e il file logico si deve scrivere:

```
PRINT#10
CLOSE10
```

L'esempio seguente mostra come produrre un listato di programma, usando il comando LIST nel programma stesso.

```
1 REM SP30
10 OPEN10,4
20 CMD10,"COMMODORE"
30 LIST
```

Dopo aver ottenuto il listato si devono scrivere in modo immediato i comandi che seguono:

```
PRINT#10
CLOSE10
```

per ottenere di chiudere la comunicazione con la stampante. Infatti dopo la LIST il controllo non torna più al programma. Nel seguito, quando si riportano esempi in cui, dopo l'esecuzione, si stampa la lista, non si ripeterà la nota per la chiusura della linea.

Esistono dei codici di controllo particolari che consentono diverse variazioni sul modo di stampare. Essi sono elencati con il codice, che deve comparire come argomento della funzione CHR$ nei comandi di PRINT:

. 15 carattere standard
. 14 carattere doppia ampiezza
. 8 grafico
. 26 ripetizione caratteri grafici
. 16 posizione inizio stampa carattere
. 27 posizione di inizio stampa con indirizz. a punti
.145 CURSOR UP (set maiuscolo/grafico)
. 17 CURSOR DOWN (set maiuscolo/minuscolo)
. 18 campo inverso
.146 annullamento campo inverso
. 10 interlinea (line feed)
. 13 RETURN

Modo carattere standard: 15
E', come dice il nome, il modo in cui opera normalmente la stampante, quando viene accesa. Dopo un cambiamento di modo si deve usare questo codice per tornare al modo normale.

Modo carattere doppia ampiezza: 14

Il programma SP40 produce una intestazione con caratteri a doppia ampiezza, seguita dal listato del programma con caratteri standard.

COMMODORE

```
1 REM SP40
10 OPEN10,4
15 CMD10
20 PRINTCHR$(14)"COMMODORE"
30 PRINTCHR$(15)
40 LIST
```

Modo grafico: 8
Si può ottenere la stampa di caratteri particolari, detti grafici, definiti dall'utente su una matrice di punti. La procedura da seguire è spiegata nel caso di un carattere particolare, disegnato di seguito, che occupa una matrice 7x6.

1	0 0 1 1 0 0	$2^0 = 1$
2	0 1 1 1 1 0	$2^1 = 2$
3	1 1 0 0 1 1	$2^2 = 4$
4	1 0 0 0 0 1	$2^3 = 8$
5	1 1 0 0 1 1	$2^4 = 16$
6	0 1 1 1 1 0	$2^5 = 32$
7	0 0 1 1 0 0	$2^6 = 64$

Nella precedente matrice, di 7 righe e 6 colonne, sono stati segnati con 1 i punti da disegnare. Ad ogni riga della matrice corrisponde una potenza del 2, crescente dall'alto verso il basso, da 0 a 6. Dopo aver disegnato la matrice, si deve scrivere sotto ogni colonna la somma dei bit 1 moltiplicati per la potenza del 2 corrispondente, e aggiungere 128. Nell'esempio si ottiene:

156, 182, 227, 227, 182, 156

Questi 6 numeri devono essere introdotti nel programma come stringa, uno dopo l'altro. Segue il programma esemplificativo SP50.

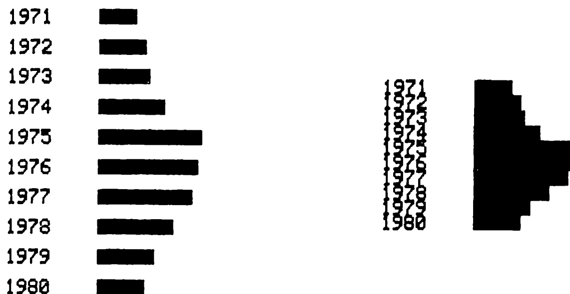
```
1 REMSP50
10 OPEN10,4
15 CMD10:A$=""
20 DATA156,182,227,227,182,156
30 FORI=1TO6
40 READA
45 A$=A$+CHR$(A)
50 NEXTI
70 PRINTCHR$(8)A$A$A$A$A$
75 PRINT:PRINTCHR$(15)
80 LIST
```

Nel programma i valori del carattere grafico sono scritti in una DATA; essi poi vengono ammucchiati nella stringa A$, che viene stampata 5 volte dopo il carattere di controllo 8. Si vede come risultato la stampa di 5 caratteri grafici speciali uno vicino all'altro. Si noti l'uso del codice 15 per tornare al modo normale prima del comando LIST.

Modo ripetizione caratteri grafici: 26
Nel programma SP60 si stampa un istogramma; il codice 26 fa ripetere, per il numero di volte indicato nel codice che segue, la prima colonna del carattere grafico definito.

```
1 REM SP60
10 OPEN10,4
15 CMD10:A$=""
20 DATA255,255,255,255,255,255
25 DATA15,20,22,30,50,48,45,35,25,20
30 FORI=1TO6
40 READA
45 A$=A$+CHR$(A)
50 NEXTI
70 FORI=1TO10
80 READB
90 C=1970+I
100 PRINTCHR$(15)C;"    ";CHR$(8)CHR$(26)CHR$(B)A$:PRINT
110 NEXTI
115 PRINT:PRINTCHR$(15)
120 LIST
```

I risultati del programma sono i seguenti (nella parte sinistra si vedono i risultati del programma come è stato listato, nella parte destra gli stessi risultati togliendo la PRINT finale alla linea 100):



Modo posizione di inizio stampa: 16
Il codice 16 deve essere seguito dal numero di colonna in cui si vuole stampare, dato come 2 codici ASCII delle 2 cifre che lo compongono (nell'esempio, per avere colonna 35 si usano i codici 51 e 53). Nel programma SP70 si mostra un esempio.

```
1 REM SP70
10 OPEN10,4
25 CMD10:A$=""
30 DATA156,182,227,227,182,156
35 FORI=1TO6
40 READA
50 A$=A$+CHR$(A)
60 NEXT
70 PRINTCHR$(8)CHR$(16)CHR$(51)CHR$(53)CHR$(26)CHR$(10)A$
75 PRINTCHR$(15):PRINT
80 LIST
```

Nell'esempio si usa il carattere grafico definito prima, producendo 10 volte la prima colonna e poi le restanti 5 colonne.



Modo posizione di inizio stampa a punti: 27
In questo caso si deve usare il codice 27 seguito dal codice 16 e subito dopo da 2 codici che danno il punto di inizio tra i 480 possibili. I due codici per il punto si ottengono considerando i due byte seguenti:

primo byte	secondo byte
0 0 0 0 0 0 0 D8	D7 D6 D5 D4 D3 D2 D1 D0

dove vengono utilizzati 9 bit. D8 si ottiene dividendo per 256 il numero che rappresenta la posizione del punto di inizio stampa, e prendendo la parte intera del risultato, che può essere 0 o 1; il byte meno significativo, invece, contiene il resto della divisione. Nel programma SP80 che segue viene stampata una sbarretta verticale nelle posizioni multiple di 50 nella riga del foglio.

```
        |        |        |        |        |        |        |

1 REM SP80
10 OPEN10,4
15 CMD10
20 FORI=1TO9
30 A=50*I
40 B=INT(A/256)
50 C=A-B*256
60 PRINTCHR$(8)CHR$(27)CHR$(16)CHR$(B)CHR$(C)CHR$(255);
70 NEXTI:PRINTCHR$(15)
80 LIST
```

Modo CURSOR UP: 145
Selezione il set dei caratteri grafico. È il modo in cui lavora la stampante se nella OPEN sa = 0 o viene omesso.

Modo CURSOR DOWN: 17
Seleziona il set maiuscolo/minuscolo. Questo effetto è ottenuto anche usando sa=7 nella OPEN. Nel programma SP90 che segue si esemplifica questo usando la OPEN con 7 e il codice 17 nella PRINT.

```
commodore


1 rem sP90
10 oPen10,4,7
20 cmd10
30 Printchr$(17)"commodore"
40 Printchr$(15):Print
50 list
```

Nelle due versioni seguenti si usa in alternativa o la OPEN con sa = 7 o solo il codice 17 ottenendo gli stessi risultati

```
commodore


1 rem sP90
10 oPen10,4,7
20 cmd10
30 Print"commodore"
40 Printchr$(15):Print
50 list
```

```
commodore


1 rem sP90
10 oPen10,4,0
20 cmd10
30 Printchr$(17)"commodore"
40 Printchr$(15):Print
50 list
```

Modo campo inverso: 18
Produce la stampa in campo inverso.

Modo annullamento campo inverso: 146
Annulla l'effetto del codice precedente.

Modo interlinea e a capo: 10 e 13
Producono un salto a nuova linea.

Nel programma SP100 che segue si dà una dimostrazione dell'effetto di questi codici.

```
1 REM SP100
10 OPEN129,4
20 PRINT#129,"PROVA1 LFN>128"CHR$(10)"PROVA DI CHR$(10)"
30 PRINT#129,"PROVA2 LFN>128"CHR$(13)"PROVA DI CHR$(13)"
40 CLOSE129
50 OPEN10,4
60 PRINT#10,"PROVA3 LFN<128"CHR$(10)"PROVA DI CHR$(10)"
70 PRINT#10,"PROVA4 LFN<128"CHR$(13)"PROVA DI CHR$(13)"
80 CLOSE10
90 CLOSE129
```

Ecco i risultati del programma precedente.

```
PROVA1 LFN>128
PROVA DI CHR$(10)

PROVA2 LFN>128
PROVA DI CHR$(13)

PROVA3 LFN<128
PROVA DI CHR$(10)
PROVA4 LFN<128
PROVA DI CHR$(13)
```

Si fa notare come, con lfn>128, si ottiene alla fine della linea di stampa un RETURN e un LINE FEED.

4.4 STAMPA AUTOMATICA

La stampante contiene al suo interno un buffer, che memorizza i dati da stampare, nonchè gli eventuali caratteri di controllo, fino a un massimo di 90 byte. Quando il buffer è pieno, esso viene automaticamente svuotato con una stampa, evitando così di perdere dati. Per questa ragione la dimensione del buffer non rappresenta un problema per l'utente; queste operazioni risultano trasparenti per l'utente che deve solo occuparsi delle righe di stampa che vuole ottenere.
Si deve tenere presente che:

.in una riga stanno al massimo 80 caratteri,
.il carattere CHR$(13) (RETURN) fa andare a capo,
.il carattere CHR$(10) (LINE FEED) fa saltare una linea; in realtà, su questo particolare tipo di stampante, i codici 13 e 10 hanno lo stesso effetto, cioè mandano a capo a una nuova linea;

.se il comando PRINT non termina con ";" o "," viene automaticamente aggiunto un RETURN,

.la virgola fa avanzare fino al completamento a 16 caratteri dell'ultimo campo stampato (o multipli di 16, contrariamente al video che usa il modulo 11),

.il punto e virgola non fa spaziare.

L'utente agisce sul buffer della stampante inviandovi dati per mezzo del comando PRINT diretto alla stampante. La stampa automatica, con conseguente svuotamento totale o parziale del buffer, avviene sotto 3 condizioni, con effetto leggermente diverso:

a)Il buffer è pieno, cioè contiene 90 caratteri tra caratteri da stampare e caratteri di controllo, ma i caratteri da stampare non raggiungono gli 80 (o i 480 punti, se si è in modo grafico). Si ha la stampa di una linea fino al totale svuotamento del buffer senza andata a capo (se non è stato incontrato un carattere di controllo per andare a capo). La prossima stampa inizierà dalla posizione dove si è arrivati.

b)Il buffer si riempie di dati che, stampati, supererebbero gli 80 caratteri. Si ha la stampa di 80 caratteri con esecuzione di un a capo e il buffer viene svuotato dei caratteri che sono stati stampati. I caratteri rimasti nel buffer verranno stampati con la stampa successiva.

c)Il buffer, pur non essendo pieno, contiene più di 80 caratteri da stampare e riceve il carattere per andare a capo. Viene stampata una prima linea più una seconda linea e si ha un a capo.

Segue il programma PROVE BUFFER PRINTER che esemplifica quanto sopra esposto.

```
1 REM SP110
2 REM PROVE BUFFER PRINTER
10 OPEN4,4
15 LF$=CHR$(10):CR$=CHR$(13)
20 A$="0123456789":B$=""
25 A1$="ABCDEFGHIJKLMNOPQRSTUVWXYZ":B1$=""
30 FORK=1TO8:B$=B$+A$:NEXTK
35 FORK=1TO4:B1$=B1$+A1$:NEXTK
40 PRINT#4,B$:GOSUB1000
50 PRINT#4,B$;A$:GOSUB1000
60 PRINT#4,B$;A$;:GOSUB1000
70 PRINT#4,LF$;LF$;LF$;LF$;LF$;LF$;LF$;LF$;LEFT$(B$,70);CR$;
75 PRINT#4,CR$;CR$;CR$;CR$;CR$;CR$;CR$;CR$;CR$;CR$;CR$;CR$;A1$:GOSUB1000
80 PRINT#4,LF$;LF$;LF$;LF$;LF$;LF$;LF$;LF$;LEFT$(B1$,70);CR$;
85 PRINT#4,CR$;CR$;CR$;CR$;CR$;CR$;CR$;CR$;CR$;CR$;CR$;CR$;A$;:GOSUB1000
999 CLOSE4:STOP
1000 FORI=1TO1000:NEXTI:RETURN
```

In questo programma vengono preparate le seguenti stringhe:
A$ con le 10 cifre decimali
B$ con le 10 cifre ripetute 8 volte,per un totale di 80 caratteri
A1$ con le 26 lettere dell'alfabeto
B1$ con le 26 lettere ripetute 4 volte, per un totale di 104 caratteri
LF$ con il carattere LINE FEED
CR$ con il carattere RETURN.
Il sottoprogramma alla linea 1000 serve per creare un breve ciclo di attesa e consentire di guardare cosa è successo sulla stampante. Alla linea 40 viene stampato B$ senza virgola o punto e virgola finale e si ha subito la stampa della riga con a capo e con svuotamento totale del buffer. Alla linea 50 si ha il riempimento del buffer con 90 caratteri da stampare, senza caratteri di controllo intermedi, ma senza virgola o punto e virgola finali, e si ottengono due righe di stampa, una completa e l'altra no, con a capo e svuotamento totale del buffer. Alla linea 60, invece, che termina con punto e virgola, si ha solo la stampa di una riga di 80 caratteri, con un a capo, e restano nel buffer i 10 caratteri di A$. La linea 70 aggiunge al buffer altri 79 caratteri (8 LF$, 70 caratteri di B$, 1 CR$), raggiungendo così gli 89 caratteri. La linea 75 continua ad aggiungere caratteri al buffer, che viene così riempito, e quindi si svuota producendo la stampa dei 10 caratteri iniziali che erano rimasti, eseguendo i LINE FEED, stampando i primi 70 caratteri di B$, eseguendo i RETURN e infine stampando A1$, dato che la linea termina senza virgola o punto e virgola. A questo punto il buffer è vuoto; esso si riempie di nuovo con i comandi delle linee 80 e 85 e riprende il meccanismo di stampa visto sopra , con la differenza che, dato che la linea 85 termina con punto e virgola e, a quel punto, il buffer non è in nessuna delle condizioni che producono stampa automatica, gli ultimi 10 caratteri di A$ restano non stampati e il programma termina. Per ottenere la stampa degli ultimi 10 caratteri si deve scrivere in immediato:

```
OPEN4,4:PRINT#4:CLOSE4
```

e così si termina.

4.5 INCOLONNAMENTI

Nella preparazione di stampati si presentano dei problemi di incolonnamento dei dati, sia numerici che alfanumerici. Il sistema trasforma tutto quello che stampa in stringhe di caratteri, lasciando le stringhe come sono, ma trasformando i dati numerici in stringa con l'operazione implicita di STR$ che aggiunge il segno o uno spazio prima del numero, e uno spazio dopo. Seguono alcuni programmi esempio che mostrano come si deve procedere per ottenere allineamenti a destra o a sinistra, tenendo conto del numero dei caratteri da stampare, e usando le funzioni TAB e

SPC. Queste funzioni hanno il seguente comportamento: TAB(n) avanza di n spazi dopo l'ultimo carattere stampato, SPC(n) stampa n spazi dopo l'ultimo carattere stampato.

```
1 REM INC1
10 OPEN4,4:CMD4
20 A$="0123456789":B$=""
30 FORK=1TO8:B$=B$+A$:NEXTK
40 PRINTB$
50 PRINTTAB(0)"POS0";TAB(6)"POS10";TAB(55)"POS70"
60 PRINTTAB(80);"NUOVA RIGA"
100 PRINT#4:CLOSE4:STOP
```

Risultati del programma INC1:

```
RISULTATI INC1

01234567890123456789012345678901234567890123456789012345678901234567890123456789
POS0      POS10                                                          POS70

NUOVA RIGA
```

In INC1 viene preparata una stringa di 80 caratteri numerici ripetendo le cifre, da 0 a 9, 8 volte; essa serve per contare le posizioni di stampa. Dopo aver stampato la stringa di controllo, alle linee 50 e 60 si usa la funzione TAB per spostarsi di un certo numero di caratteri e stampare nelle posizioni volute. Si noti che TAB(80) fa andare a capo di una linea, mentre TAB(0) fa stampare nella prima posizione.

```
1 REM INC2
10 OPEN4,4:CMD4
20 A$="0123456789":B$=""
30 FORK=1TO8:B$=B$+A$:NEXTK
40 PRINTB$
50 C$="DIECI CAR."
60 PRINTC$;SPC(10);C$
70 PRINTC$TAB(10)C$
100 PRINT#4:CLOSE4:STOP
```

Risultati del programma INC2:

```
01234567890123456789012345678901234567890123456789012345678901234567890123456789
DIECI CAR.          DIECI CAR.
DIECI CAR.          DIECI CAR.
```

In INC2 si dimostra l'uso di SPC.

```
1 REM INC3
5 DIMC(10),C$(10):SP$="          "
10 OPEN4,4:CMD4
20 A$="0123456789":B$=""
30 FORK=1TO8:B$=B$+A$:NEXTK
40 PRINTB$
50 DATA1234,8976,3456,13567,45,890,5432,9876,3456,12345
60 FORK=1TO10:READC(K):C$(K)=STR$(C(K)):NEXTK
65 FORJ=0TO1:FORK=1TO5:PRINTC(K+J*5);"   ";:NEXTK:PRINT:NEXTJ
67 PRINT:PRINTB$:PRINT
70 PRINT:PRINT"TABELLA 5X5 NUMERI IN COLONNA":PRINT
80 FORJ=0TO1:FORK=1TO5:PRINTRIGHT$(SP$+C$(K+J*5),10);:NEXTK:PRINT:NEXTJ
100 PRINT#4:CLOSE4:STOP
```

Risultati del programma INC3

```
01234567890123456789012345678901234567890123456789012345678901234567890123456789
 1234    8976    3456    13567    45
 890    5432    9876    3456    12345

01234567890123456789012345678901234567890123456789012345678901234567890123456789


TABELLA 5X5 NUMERI IN COLONNA

      1234      8976      3456     13567        45
       890      5432      9876      3456     12345
```

In INC3 si stampa dapprima una tabella di 10 numeri, 5 per riga, senza preoccuparsi degli incolonnamenti; poi si sistemano le cose e si ha la stampa con gli allineamenti a destra. I numeri vengono trasformati in stringhe e viene aggiunta a sinistra una stringa di spazi; operando con la funzione RIGHT$ si stampano tutte stringhe di uguale lunghezza (10 caratteri).

```
1 REM INC4
5 DIMC(10),C$(10):SP$="          "
10 OPEN4,4:CMD4
20 A$="0123456789":B$=""
30 FORK=1TO8:B$=B$+A$:NEXTK
40 PRINTB$
50 DATA1234,8976,3456,13567,45,890,5432,9876,3456,12345
60 FORK=1TO10:READC(K):C$(K)=STR$(C(K)):NEXTK
65 FORJ=0TO1:FORK=1TO5:PRINTC(K+J*5);"   ";:NEXTK:PRINT:NEXTJ
67 PRINT:PRINTB$:PRINT
70 PRINT:PRINT"TABELLA 5X5 NUMERI IN COLONNA":PRINT
80 FORJ=0TO1:FORK=1TO5:PRINTSPC(10-LEN(C$(K+J*5)))C$(K+J*5);:NEXTK:PRINT:NEXTJ
100 PRINT#4:CLOSE4:STOP
```

Risultati del programma INC4:

```
0123456789012345678901234567890123456789012345678901234567890123456789012345678
 1234    8976     3456     13567     45
 890     5432     9876     3456     12345

0123456789012345678901234567890123456789012345678901234567890123456789012345678


TABELLA 5X5 NUMERI IN COLONNA

       1234      8976      3456     13567        45
        890      5432      9876      3456     12345
```

In INC4 si ottengono gli stessi risultati di INC3, ma, usando, invece che la funzione RIGHT$, la funzione LEN e la funzione SPC per aggiungere ogni volta a sinistra gli spazi necessari in base alla lunghezza del numero.

```
1 REM INC5
5 DIMC$(12)
10 OPEN4,4:CMD4
20 A$="0123456789":B$=""
30 FORK=1TO8:B$=B$+A$:NEXTK
40 PRINTB$
50 DATA BELLO,BENISSIMO,ALLEGRO,PIACEVOLE,RILASSANTE,SOLEGGIATO
51 DATAGRADEVOLE,UTILE,DEFINITIVO,ARIOSO,STUPENDO,MAGNIFICO
60 FORK=1TO12:READC$(K):NEXTK
65 FORJ=0TO1:FORK=1TO5:PRINTC$(K+J*5);"  ";:NEXTK:PRINT:NEXTJ
70 N=0:FORK=1TO12:IFLEN(C$(K))>NTHENN=LEN(C$(K))
71 NEXTK
75 PRINT:PRINT
80 N=N+3:FORJ=0TO1:FORK=1TO5:PRINTC$(K+J*5);SPC(N-LEN(C$(K+J*5)));:NEXTK:PRINT:N
EXTJ
100 PRINT#4:CLOSE4:STOP
```

Risultati del programma INC5:

```
0123456789012345678901234567890123456789012345678901234567890123456789012345678
BELLO  BENISSIMO  ALLEGRO  PIACEVOLE  RILASSANTE
SOLEGGIATO  GRADEVOLE  UTILE  DEFINITIVO  ARIOSO


BELLO         BENISSIMO     ALLEGRO       PIACEVOLE     RILASSANTE
SOLEGGIATO    GRADEVOLE     UTILE         DEFINITIVO    ARIOSO
```

In INC5 si opera su una tabella di parole e si ottiene l'allineamento a sinistra sfruttando le funzioni LEN e SPC.

4.6 HARD COPY

Segue il programma COPIAVIDEO che, usando la routine scritta da 60900 a 60914, fa la copia del video sulla stampante. Il programma è costituito da sole 3 righe che lanciano la routine. La routine può essere incorporata in qualunque

programma e funziona con le due possibili posizioni della mappa video, in base al contenuto del byte 648. Le variabili usate cominciano tutte con H.

```
1 REM COPIAVIDEO
10 GOSUB60900
11 STOP
60900 REM HARD COPY
60901 H1$=CHR$(145):OPEN4,4:PRINT#4
60902 H1=7658:IFPEEK(648)=16THENH1=4074
60903 FORH0=0TO22:H0$=H1$:H1=H1+22
60904 FORH2=H1TOH1+21:H3=PEEK(H2)
60905 IFH3>128THENH3=H3-128:H4=1:H0$=H0$+CHR$(18)
60906 IF(H3>0)*(H3<32)THENH3=H3+64:GOTO60910
60907 IF(H3>31)*(H3<64)THEN60910
60908 IF(H3>63)*(H3<96)THENH3=H3+128:GOTO60910
60909 IF(H3>95)*(H3<128)THENH3=H3+64:GOTO60910
60910 H0$=H0$+CHR$(H3)
60911 IFH4=1THENH0$=H0$+CHR$(146):H4=0
60912 NEXTH2:PRINT#4,H0$:NEXTH0
60913 PRINT#4:CLOSE4
60914 RETURN
```

CAPITOLO 5

FILE SU CASSETTA

5.1 INTRODUZIONE

I file su cassetta sono la registrazione di insiemi di caratteri su nastro magnetico. Al momento della registrazione può essere assegnato un nome di riconoscimento, che viene registrato, insieme ad alcuni caratteri di controllo, prima dei caratteri che costituiscono l'insieme dei dati da registrare. Tale nome può essere usato pei andare a ricercare una particolare registrazione su un nastro che ne contenga diverse. I caratteri di controllo sono comunque sempre presenti e pensa il sistema a gestirli, cioè sono trasparenti per l'utente.

Il VIC20 può trattare due tipi diversi di file:
.1)I file ASCII, nei quali i dati sono registrati carattere per carattere a pezzi, organizzati in gruppi di caratteri separati da un "carattere separatore". In conseguenza i dati possono essere riletti o carattere per carattere o a gruppi di caratteri, e il nastro viene avviato ed arrestato a seconda delle necessità.
.2)I file BINARI, nei quali i caratteri sono registrati in modo apparentemente continuo, tutti insieme. In conseguenza questi file possono essere riletti tutti in blocco, senza arresti intermedi.

Il sistema gestisce questi file con istruzioni diverse. Per il tipo 1), che riguarda i file di dati gestiti da programma sotto forma di variabili, sono disponibili i comandi: OPEN, CLOSE, INPUT, GET e PRINT. Per il tipo 2), che riguarda i programmi, sono disponibili i comandi SAVE e LOAD, che possono essere usati sia in modo immediato che da programma.

Le registrazioni di informazioni sui supporti, magnetici o non magnetici, hanno sempre due aspetti; uno logico, relativo alle necessità dell'utente, ed uno fisico, relativo alle caratteristiche del supporto. Su un foglio di carta entrano solo un determinato numero di caratteri per riga, come sul video. Cose analoghe succedono con i nastri magnetici o i dischi. Si parla quindi di RECORD LOGICO come di un insieme di informazioni significativo per l'utente, e di RECORD FISICO o BLOCCO FISICO come del gruppo di byte che fisicamente entrano o escono dalla

memoria centrale, in una singola operazione di trasferimento dati con la memoria di massa.
Tutte le registrazioni su cassetta vengono gestite allo stesso modo per quanto riguarda il blocco fisico dei dati; il sistema usa un buffer di memoria costituito da 192 byte e situato nelle locazioni di memoria che vanno da 828 a 1020 (033CH - 03FCH). In una operazione di lettura o scrittura di blocco fisico di nastro sono coinvolti 192 byte; dopo questo blocco si ha un GAP, cioè un pezzetto di nastro inutilizzato. Questo pezzetto di nastro è quello che viene svolto durante il tempo di frenata e arresto del movimento del nastro. Per ragioni di sicurezza ogni blocco fisico di nastro viene registrato 2 volte e in conseguenza letto 2 volte, anche se l'utente non se ne accorge.
Si ricordano alcune locazioni della pagina zero della memoria coinvolte nelle operazioni su cassetta:

.166 (00A6H)	puntatore al buffer
.172 e 173 (00ACH-00ADH)	indirizzo iniziale di memorizzazione per i file binari
.174 e 175 (00AEH-00AFH)	indirizzo finale di memorizzazione per i file binari
.178 e 179 (00B2H-00B3H)	indirizzo inizio buffer cassetta
.183 (00B7H)	lunghezza in caratteri del nome del file in uso (massimo 128 caratteri)
.184 (00B8H)	numero logico del file in uso
.185 (00B9H)	indirizzo secondario operazione in corso
.186 (00BAH)	numero della periferica attiva
.187 e 188 (00BBH-00BCH)	indirizzo della stringa che contiene il nome del file in uso
.601/610 (0259H/0262H)	tabella dei numeri logici dei file
.611/620 (0263K/026CH)	tabella dei numeri delle periferiche
.621/630 (026DH/0276H)	tabella indirizzi secondari
.828/1020 (033CH/03FCH)	buffer cassetta di 192 caratteri

Si noti che la tabella dei numeri logici dei file è usata anche per i file che non si trovano su cassetta.

5.2 FILE DI PROGRAMMI

Per memorizzare un programma, presente in memoria, su nastro basta scrivere:

SAVE "nome" oppure

SAVE "nome",1

Il numero 1, che è il numero del registratore a cassetta, può essere sottinteso. Il sistema risponde con un messaggio sul video che chiede di avviare il nastro per registrare:

PRESS RECORD & PLAY ON TAPE

si deve fare attenzione e premere i due tasti nel modo corretto: prima il tasto RECORD e poi il tasto PLAY. Questa procedura è necessaria perchè le operazioni di trasmissione dati da memoria a cassetta iniziano dal momento in cui viene premuto il PLAY; se questo è premuto prima di RECORD si rischia di perdere una parte della registrazione. Durante la registrazione compaiono alcuni messaggi sul video e a registrazione ultimata il registratore si ferma segnalando ancora con un messaggio la fine dell'operazione. A questo punto è consigliabile disabilitare la registrazione, riavvolgere il nastro e scrivere:

VERIFY "nome" oppure

VERIFY "nome",1

appare il messaggio:

PRESS PLAY ON TAPE

Quando si preme il tasto PLAY, viene confrontata la registrazione su nastro con il contenuto della memoria; se non ci sono differenze, sul video compare il messaggio O.K. Se non compare, si deve ripetere la procedura di registrazione e verifica.

Il nome del programma può essere lungo fino a 128 caratteri. La memorizzazione avviene almeno su 3 blocchi fisici. Un primo blocco reca il nome del programma e alcune indicazioni sulla sua memorizzazione; seguono uno o più blocchi, a seconda della lunghezza del programma, e infine un blocco di chiusura. In questo conteggio non si fa riferimento alla doppia memorizzazione. Il sistema opera così: riempie il buffer con i caratteri che ci stanno, quando è pieno lo scrive, poi lo riempie nuovamente e continua così fino alla fine. Il programma viene registrato in modo che ogni parola chiave del Basic dà luogo a un solo byte (token), che è il codice della stessa (si veda l'appendice J del primo volume), esattamente come è registrato nella

memoria centrale. Si ha praticamente un travaso di byte tra memoria e cassetta, tramite il buffer.

Segue il programma CASS6, il quale contiene solo una routine per stampare il contenuto del buffer della cassetta sulla stampante. Per provarlo si deve procedere così: scrivere il programma e salvarlo su cassetta con il comando: SAVE "CASS6". Dare il RUN: si vedrà stampato il contenuto del buffer dopo l'esecuzione del comando SAVE. Si vedrà che il byte 830 contiene l'indirizzo precedente all'inizio del programma Basic (nell'esempio 16*256=4096, VIC senza espansioni) e che i byte 831 e 832 contengono l'indirizzo di fine del programma (nel caso 16*256+188=4284). Se si vanno a verificare i contenuti dei byte con delle PEEK si vedrà che la locazione 4280 contiene 142, il codice di RETURN, il byte seguente è zero e così pure i due successivi, per segnalare la fine del programma; in 4284 comincia la zona variabili.

```
1 REM CASS6
10 REM PROVA SAVE PROGRAMMA
50 OPEN4,4:GOSUB1000
60 CLOSE4:STOP
999 REM ROUTINE STAMPA BUFFER
1000 I=828
1010 FORK=1TO48:FORL=1TO4
1020 PRINT#4,I;" ";MID$(STR$(PEEK(I))+"   ",2,3);
1030 I=I+1:NEXTL:PRINT#4
1040 NEXTK:PRINT#4:PRINT#4:RETURN
```

Risultati del programma CASS6, prima parte:

828	1	829	1	830	16	831	188
832	16	833	67	834	65	835	83
836	83	837	54	838	32	839	32
840	32	841	32	842	32	843	32
844	32	845	32	846	32	847	32
848	32	849	32	850	32	851	32
852	32	853	32	854	32	855	32
856	32	857	32	858	32	859	32
860	32	861	32	862	32	863	32
864	32	865	32	866	32	867	32
868	32	869	32	870	32	871	32
872	32	873	32	874	32	875	32
876	32	877	32	878	32	879	32
880	32	881	32	882	32	883	32
884	32	885	32	886	32	887	32
888	32	889	32	890	32	891	32
892	32	893	32	894	32	895	32
896	32	897	32	898	32	899	32
900	32	901	32	902	32	903	32
904	32	905	32	906	32	907	32
908	32	909	32	910	32	911	32
912	32	913	32	914	32	915	32
916	32	917	32	918	32	919	32
920	32	921	32	922	32	923	32

924	32	925	32	926	32	927	32
928	32	929	32	930	32	931	32
932	32	933	32	934	32	935	32
936	32	937	32	938	32	939	32
940	32	941	32	942	32	943	32
944	32	945	32	946	32	947	32
948	32	949	32	950	32	951	32
952	32	953	32	954	32	955	32
956	32	957	32	958	32	959	32
960	32	961	32	962	32	963	32
964	32	965	32	966	32	967	32
968	32	969	32	970	32	971	32
972	32	973	32	974	32	975	32
976	32	977	32	978	32	979	32
980	32	981	32	982	32	983	32
984	32	985	32	986	32	987	32
988	32	989	32	990	32	991	32
992	32	993	32	994	32	995	32
996	32	997	32	998	32	999	32
1000	32	1001	32	1002	32	1003	32
1004	32	1005	32	1006	32	1007	32
1008	32	1009	32	1010	32	1011	32
1012	32	1013	32	1014	32	1015	32
1016	32	1017	32	1018	32	1019	32

Si possono modificare le uscite e produrle sul video se non si dispone di una stampante.
Si possono verificare, sempre usando la funzione PEEK, i contenuti di alcuni dei puntatori citati.

Per caricare i programmi da nastro si usa il comando:

LOAD “nome” oppure

LOAD “nome”,1

sul video appare il messaggio:

PRESS PLAY ON TAPE

e vengono comunicati i nomi dei file incontrati prima di quello richiesto. Se non si usa il nome, viene caricato in memoria il primo programma che si trova su nastro. Durante il caricamento appaiono dei messaggi sul video, fino al READY finale, che compare se il caricamento è avvenuto correttamente. Se si ha un messaggio di errore si può ritentare il caricamento, ma in caso di errore persistente è probabile che il nastro sia danneggiato.
Si raccomanda di proteggere le cassette registrate togliendo le apposite linguette.

Si rimanda all'Appendice E relativa al cartridge PROGRAMMER'S AID e al paragrafo 8.2 su VICMON per ampliamenti sul trattamento dei file di programmi.

Segue il programma CASS7, nel quale un file di programma, CASS7 stesso, viene aperto come file di dati per leggerlo, e vengono stampati i contenuti del buffer dopo la OPEN, dopo la GET (il buffer viene riempito con la prima GET) e dopo aver eseguito altre 150 GET, per ottenere un nuovo riempimento del buffer (50+150=200 > 192). Dalla stampa si vede che dopo la OPEN nel buffer c'è il nome del file con i caratteri di controllo già visti. Dopo la GET iniziale si vedono le istruzioni del programma, mentre dopo le altre 199 GET nel buffer c'è una parte del programma che si trova più avanti del previsto; infatti il GAP nei file di programma è più corto di quello dei file di dati e quindi viene saltata una parte di programma.

```
1 REM CASS7
10 REM PROVA BUFFER PROGRAMMA
50 OPEN4,4:PRINT#4,"BUFFER DOPO OPEN":PRINT#4
51 OPEN1,1,0:GOSUB1000:PRINT#4:PRINT#4
52 PRINT#4,"50 CARATTERI LETTI CON GET":PRINT#4
53 FORK=1TO50:GET#1,B$:PRINT#4,B$;:NEXTK:PRINT#4:PRINT#4
55 PRINT#4,"BUFFER BLOCCO PROGRAMMA":PRINT#4
57 GOSUB1000:PRINT#4:PRINT#4
59 FORK=1TO150:GET#1,B$:PRINTK;" ";:NEXTK
60 PRINT#4,"BUFFER ULTIMO BLOCCO":PRINT#4
```

```
65 GOSUB1000:PRINT#4:PRINT#4
70 CLOSE4:STOP
999 REM ROUTINE STAMPA BUFFER
1000 I=828
1010 FORK=1TO48:FORL=1TO4
1020 PRINT#4,I;" ";MID$(STR$(PEEK(I))+"  ",2,3);
1030 I=I+1:NEXTL:PRINT#4
1040 NEXTK:PRINT#4:PRINT#4:RETURN
```

Risultati del programma CASS7:

```
50 CARATTERI LETTI CON GET

                              CASS7# PROVA BUFFER PROGRAMMAL4,4:
```

BUFFER BLOCCO PROGRAMMA

828	13	829	16	830	1	831	0
832	143	833	32	834	67	835	65
836	83	837	83	838	55	839	0
840	42	841	16	842	10	843	0
844	143	845	32	846	80	847	82
848	79	849	86	850	65	851	32
852	66	853	85	854	70	855	70
856	69	857	82	858	32	859	80
860	82	861	79	862	71	863	82
864	65	865	77	866	77	867	65
868	0	869	76	870	16	871	50
872	0	873	159	874	52	875	44
876	52	877	58	878	152	879	52
880	44	881	34	882	66	883	85
884	70	885	70	886	69	887	82
888	32	889	68	890	79	891	80
892	79	893	32	894	79	895	80
896	69	897	78	898	34	899	58
900	152	901	52	902	0	903	99
904	16	905	51	906	0	907	[illegible]
908	49	909	44	910	[illegible]	[illegible]	
[illegible]							

BUFFER ULTIMO BLOCCO

828	13	829	16	830	1	831	0
832	143	833	32	834	67	835	65
836	83	837	83	838	55	839	0
840	42	841	16	842	10	843	0
844	143	845	32	846	80	847	82
848	79	849	86	850	65	851	32
852	66	853	85	854	70	855	70
856	69	857	82	858	32	859	80
860	82	861	79	862	71	863	82
864	65	865	77	866	77	867	65
868	0	869	76	870	16	871	50
872	0	873	159	874	52	875	44
876	52	877	58	878	152	879	52
880	44	881	34	882	66	883	85
884	70	885	70	886	69	887	82
888	32	889	68	890	79	891	80
892	79	893	32	894	79	895	80
896	69	897	78	898	34	899	58
900	152	901	52	902	0	903	99
904	16	905	51	906	0	907	159
908	49	909	44	910	49	911	44
912	48	913	[illegible]	[illegible]	[illegible]	[illegible]	49
[illegible]							[illegible]

I programmi vengono salvati conservando l'indirizzo dove erano memorizzati, però, al momento del caricamento (LOAD), vengono rilocati in memoria all'indirizzo prelevato dai due byte che puntano all'inizio del BASIC (43/44 - 002BH/002CH). Si ricorda che questi due byte possono essere manipolati dall'utente. Quando il programma viene rilocato, vengono modificati dal sistema i due byte di link di ogni istruzione.
Il comando SAVE può essere usato in una forma diversa da quella sopra esposta, per ottenere che il programma salvato su nastro non venga rilocato al momento del caricamento. Questo può essere comodo per programmi particolari che si desidera

far partire sempre da uno stesso indirizzo, anche con configurazioni diverse di memoria. Naturalmente si devono considerare tutti gli aspetti del problema. Se si carica in memoria un programma che inizia in posizione diversa da quella indicata dai puntatori all'inizio del BASIC, esso non parte; per farlo partire, o per listarlo, bisogna spostare il puntatore. Le forme possibili per il comando sono:

SAVE "nome",1,1
dove il secondo "1" evita che il programma venga rilocato in fase di caricamento, usando un normale comando di LOAD.

SAVE "nome",1,2
dove il "2" fa aggiungere dopo la registrazione del programma una segnalazione di fine nastro (END OF TAPE).

SAVE "nome",1,3
dove il "3" produce i due effetti contemporaneamente.

Analogamente si può usare una forma di LOAD che carichi un programma, salvato con un comando normale, in modo da non rilocarlo in memoria. Il comando si scrive:

LOAD "nome",1,1
e il secondo "1" non fa avvenire la rilocazione del programma.

Si raccomanda di procedere con cautela e di tener presenti tutte le caratteristiche del sistema.

Si suggerisce ora un metodo un po' "manuale" per fondere due programmi, non disponendo del cartridge apposito, (Appendice E, Programmer AID'S, funzione MERGE) oppure per fondere due programmi abbastanza lunghi e che quindi non si riesce a trattare con il comando MERGE. Si deve operare così:

.1)Si carica in memoria il più corto dei 2 programmi;

.2)Si pone nel registratore una cassetta libera e si scrivono sul video i comandi che seguono, in modo immediato:

```
OPEN1,1,1:CMD1:LISTN1-N2
PRINT#1
CLOSE1
```

si scrive la prima riga ponendo al posto di n1 il numero della prima linea da memorizzare, e al posto di n2 il numero dell'ultima linea (del programma che è in memoria), facendo in modo che il numero di linee occupi sul video, in caso di lista, circa i due terzi. Premendo RETURN si ottiene il solito messaggio di richiesta di avviamento registratore per memorizzare. Viene aperto un file senza nome e vengono registrate in modo lista, e quindi con il carattere CHR$(13) finale, le linee

indicate. Quando ricompare il cursore, si scrive la seconda riga seguita da RETURN e poi la terza per chiudere il file. Si procede così, registrando a pezzetti il programma che sta in memoria. Ogni volta basta tornare sù con il cursore e modificare n1 e n2.

.3)Si carica in memoria l'altro programma e si aggiungono come prime istruzioni o come ultime istruzioni le linee che seguono:

```
1 OPEN1:PRINT"[illegible]";
2 GET#1,A$:PRINTA$;
3 IFST=64THENCLOSE1:END
4 GOTO2
```

.4)Si riavvolge il nastro preparato precedentemente e si scrive RUN1, se la numerazione delle 4 linee aggiunte è come sopra, o altro, per far partire il programma. Questa routine apre gli spezzoni di file registrati e li lista sul video, andando a capo ad ogni CHR$(13) che incontra, poi chiude il file su cassetta e si ferma. A questo punto, riportando in alto il cursore, si confermano le linee di programma che sono listate sul video una per una; in tale modo esse vengono incorporate nel programma che è in memoria. Si procede per tutti i pezzi di programma memorizzati, dando sempre lo stesso tipo di RUN.

.5)Si memorizza il programma su nastro o su disco.

Naturalmente se i due programmi hanno linee doppie, quella confermata cancella quella già presente in memoria con lo stesso numero di linea. E' un metodo che richiede un pò di pazienza, ma si può fare!

5.3 FILE DI DATI

I file di dati sono insiemi di registrazioni logicamente suddivise in parti che prendono il nome di record. Un record comprende dati che, per qualche ragione, si ritiene utile raggruppare. Il classico esempio è quello che si riferisce ai dati anagrafici dei dipendenti di un azienda. Per il signor PIPPO PLUTO si registrano diverse informazioni come il luogo e la data di nascita, la residenza, il telefono, la composizione della famiglia, i titoli di studio, le precedenti occupazioni, la data di assunzione, le mansioni, ecc.. Per ogni dipendente si ha un record logico; le diverse voci che lo compongono si chiamano campi. Un campo può essere diviso ulteriormente in sottocampi, come per esempio l'indirizzo, che è composto da via, numero civico, CAP e località. In generale viene scelto un campo come elemento principale del record, e i record vengono mantenuti in ordine alfabetico o numerico in base al

campo prescelto che viene detto "chiave del record". In ogni lavoro che comporti elaborazioni di grandi quantità di dati, si usano file memorizzati su memorie di massa come nastri o dischi magnetici. Caratteristica dei file di dati è che si deve poter introdurre nel calcolatore solo la parte di registrazioni che interessa, e non tutto il file, come succede invece con i file di programmi.
La definizione della struttura dei file di dati è uno dei lavori più delicati nell'analisi dei problemi da risolvere con il calcolatore. File male organizzati condizionano pesantemente tutte le procedure elaborative.
Quando si definiscono i file, si deve stendere un elenco dei contenuti del record, precisando le caratteristiche di ogni campo e sottocampo, cioè dati alfabetici o numerici, lunghezza fissa o variabile, e si deve tenere conto dello spazio occupato dai caratteri separatori di campo. E' importante anche sapere quanti record devono essere trattati, per poter stabilire le dimensioni dei supporti di memoria necessari per le registrazioni. Esistono cassette da 5 minuti, 10 minuti, ecc..
Avendo come supporto un nastro magnetico su cassetta non si hanno molti gradi di libertà nelle scelte. Per prima cosa bisogna considerare che il mezzo offre solo la possibilità di registrare in modo sequenziale i record, uno dopo l'altro, e che si presta alla rielaborazione degli stessi record solo in modo sequenziale. Inoltre non si possono apportare modifiche a qualche record, facendo riscritture parziali che risultano sempre molto rischiose; per correggere un file su nastro bisogna riscriverlo tutto. Questa operazione non presenta grossi problemi se si possono collegare contemporaneamente al calcolatore 2 registratori, uno per leggere e l'altro per scrivere, cosa non possibile sul VIC20. Più problematico è invece il caso che si disponga di un solo registratore; si deve infatti allora caricare tutto il file in memoria (se ci entra), modificare i dati in memoria e poi riscrivere tutto su cassetta.

Il VIC20 consente di scrivere file sequenziali su cassetta con record di lunghezza fissa o variabile e quindi con campi di lunghezza fissa o variabile. Non si hanno limiti per le dimensioni del record logico, salvo che, per certi tipi di scrittura, il campo compreso tra 2 caratteri separatori (CHR$(13)) non può superare 87 caratteri. Il sistema gestisce i record logici in modo completamente trasparente per l'utente, li ammucchia nel buffer carattere per carattere e, quando il buffer è pieno, lo scrive; poi ricomincia a riempirlo. In questo modo i record logici e i campi possono venire spezzati, ma senza problemi per l'utente.
Con il termine "file sequenziale" si intende un file nel quale si possono effettuare solo registrazioni ed elaborazioni sequenziali, un record dopo l'altro.

Per la particolare natura del linguaggio Basic tutto quello che si scrive, anche su nastro, viene scritto carattere per carattere, anche se nelle operazioni di scrittura e lettura si usano i nomi delle variabili; il sistema trasforma poi il contenuto della variabile in una sequenza di caratteri. Praticamente succede esattamente quello che succede quando si evidenziano dati sul video.
Questo significa che non si ha una corrispondenza uno a uno tra byte di registrazione e byte di memoria; quando si leggono byte del buffer in una variabile essi vanno

ad occupare lo spazio che il Basic gli assegna (vedasi la trasformazione delle variabili numeriche).
Nelle operazioni relative ai file sono coinvolti 4 parametri; essi sono:

.lfn numero logico del file;
.dn numero della periferica;
.fn nome del file;
.sa indirizzo secondario.

Essi hanno il seguente significato:

.lfn è un numero che può andare da 1 a 255 e serve a individuare un file e a distinguerlo da altri file trattati contemporaneamente. Il sistema consente di trattare contemporaneamente fino a 3 file.

.dn è il numero della periferica ed è determinato dalla configurazione fisica del sistema. Di norma la cassetta ha dn=1.

.fn è il nome del file, con il quale viene registrato. Per la cassetta non è obbligatorio, ma è consigliabile usarlo. Tale nome non deve superare i 128 caratteri.

.sa è l'indirizzo secondario che per la cassetta serve a distinguere il tipo di operazione; si ha:
-sa=0 per operazioni di lettura;
-sa=1 per operazioni di scrittura senza segnalazione di fine nastro;
-sa=2 per operazioni di scrittura con segnalazione di fine nastro.

Gli esempi che seguono serviranno a chiarire la terminologia, che viene usata con il significato spiegato sopra.

Le istruzioni BASIC per la cassetta sono:

OPEN lfn, dn, sa, fn

Serve per aprire il file, cioè per predisporlo alla elaborazione; è obbligatorio aprire i file sia per leggere che per scrivere. Dopo questa operazione sono presenti in memoria le informazioni necessarie al sistema per gestire il file.

CLOSE lfn

Esclude il file dall'elaborazione; è obbligatorio chiudere i file. Quando si chiude un

file di scrittura viene materialmente scritto l'ultimo blocco fisico; in tutti i casi vengono tolti dalla memoria i riferimenti a quel file, che non è più gestibile, salvo riapertura.

PRINT# lfn,lista

dove:

lista è la lista delle variabili separate da un separatore

lfn è lo stesso numero logico del file usato nella OPEN

Scrive sul file i contenuti delle variabili della lista, carattere per carattere. Si deve fare attenzione al carattere separatore usato. Si può usare sia la virgola che il punto e virgola; essi hanno lo stesso effetto che sul video, cioè il punto e virgola non aggiunge spazi, mentre la virgola li aggiunge. Però questi separatori non vengono conservati nella registrazione e quindi le variabili si uniscono tra loro procurando alcune complicazioni. Per poter ritrovare le variabili come si erano definite, si devono forzare dei caratteri separatori anche sul nastro; questi sono il RETURN e la virgola, e devono essere passati come stringhe. Si deve cioè usare: CHR$(13) per RETURN e CHR$(44), oppure "," per la virgola. E' importante anche come termina la lista delle variabili; se manca un separatore, viene aggiunto il RETURN. Si consiglia quindi di usare sempre, come separatore, il punto e virgola e i separatori forzati tra le variabili, e di decidere se usarlo o meno a fine lista.
Si ricorda che il numero massimo di caratteri tra 2 RETURN deve essere 87; per questa ragione non sempre si possono usare come caratteri separatori tra le variabili le virgole forzate. Questa osservazione vale se il record deve essere letto con il comando INPUT, che legge variabili; se si usa il comando GET, che legge caratteri, i campi compresi tra due RETURN possono essere anche più lunghi di 87 caratteri.

Segue il programma CASS1, che scrive un file di dati di nome PROVACASS, composto da record aventi 3 campi ciascuno, di lunghezza variabile. I campi sono rispettivamente: COGNOME, NOME, DATO NUMERICO (letto come stringa). Si possono scrivere quanti record si vuole; per terminare si deve rispondere con 4 asterischi alla richiesta del cognome.

```
100 REM CASS1
110 REM CARICAMENTO FILE SEQUENZIALE SU CASSETTA
120 REM RECORD LOGICO FORMATO DA 3 CAMPI DI LUNGHEZZA VARIABILE
130 REM A$←COGNOME  B$←NOME  C$←DATO NUMERICO LETTO COME STRINGA
140 REM PER TERMINARE RISPONDERE AL COGNOME CON 4 ASTERISCHI
150 REM SI USA CHR$(13) COME SEPARATORE DI CAMPO
160 REM I 3 CAMPI SONO SCRITTI USANDO UN SOLO COMANDO PRINT
170 REM DOPO C$ NON SI USA CH$ DATO CHE MANCANDO LA VIRGOLA O
180 REM IL PUNTO E VIRGOLA IL SISTEMA AGGIUNGE CHR$(13)
190 REM SI USA IL ";" COME SEPARATORE PER NON SAGGIUNGERE SPAZI
200 CH$=CHR$(13):OPEN1,1,1,"PROVACASS"
210 PRINT"[CLR]PER USCIRE SCRIVERE **** PER COGNOME[DOWN][DOWN]"
```

```
220 INPUT"COGNOME: ";A$
230 IFA$="****"THEN280
240 INPUT"NOME: ";B$
250 INPUT"NUMERO:";C$
260 PRINT#1,A$;CH$;B$;CH$;C$
270 GOTO220
280 CLOSE1:STOP
```

In questo programma il record logico è formato da 3 campi, e si usa lo stesso separatore tra campi e record, il CHR$(13). Si poteva usare il separatore virgola forzata tra i campi, ma con il rischio di superare gli 87 caratteri ed avere errore. Si ricorda che quando nei comandi di INPUT si usa la frase esplicativa tra apici, questa non deve superare i 20 caratteri, altrimenti viene danneggiato il contenuto della variabile di ingresso.

Vediamo ora i comandi di lettura; essi sono:

INPUT# lfn, lista

lista è la lista variabili separate da virgola

lfn è il numero logico del file usato nella OPEN

Legge una variabile per volta, per ogni variabile della lista, prelevando i dati dal buffer della cassetta, carattere per carattere, procedendo fino al primo separatore. Le variabili possono essere di tipo stringa o di tipo numerico; in quelle di tipo stringa può venir letto qualunque carattere. Per le variabili numeriche possono esserci problemi; infatti il sistema legge carattere per carattere e poi esegue la funzone VAL di quello che ha letto. La VAL dà errore se trova caratteri diversi da cifre, dal punto decimale e dal segno.

GET# lfn,nome variabile dove:

"nome variabile" serve per ricevere il carattere.

Legge un carattere per volta. La variabile può essere stringa o numerica, con le avvertenze di prima per le numeriche. Si usa se il blocco scritto ha campi che superano gli 87 caratteri tra due RETURN.

Segue il programma CASS2 che serve per rileggere il file PROVACASS, scritto con il programma CASS1, e stamparlo sul video.

```
100 REM CASS2
110 REM LETTURA FILE PROVACASS
120 REM CON UN COMANDO DI INPUT VENGONO LETTI I 3 CAMPI DEL RECORD
130 REM LOGICO - DOPO L'INPUT VIENE MEMORIZZATO ST (FLAG DI STATO) IN TS
140 REM TS VIENE ANALIZZATO PER CONDIZIONE DI FINE FILE
150 REM PRIMA DI TORNARE A LEGGERE
160 OPEN1,1,0,"PROVACASS":PRINT"♥STAMPA FILE PROVACASSQQ"
170 INPUT#1,A$,B$,C$
180 TS=ST
190 PRINTA$,B$,C$
200 IFTS=64THENCLOSE1:STOP
210 GOTO170
```

Il file PROVACASS è stato scritto con indirizzo secondario 1 nella OPEN, quindi è stato registrato solo con il carattere finale di fine file, che è uno zero binario (tutti bit 0). Se si usasse l'indirizzo secondario 2 si avrebbe ancora lo zero binario di chiusura file, ma il sistema aggiungerebbe un ulteriore blocco fisico con la segnalazione di fine nastro.

In fase di lettura con il comando INPUT, i dati vengono prelevati dal buffer fino a quando viene incontrato uno dei due possibili caratteri separatori (RETURN o ,). Il carattere di fine file viene sentito in anticipo sulla lettura dell'ultimo dato. Per questa ragione ST deve essere memorizzato subito dopo la lettura e analizzato prima di andare ancora a leggere (ST=64 segnala la fine del file).
Se si usa il comando GET, si ottengono i caratteri uno alla volta compresi i caratteri di controllo che, con il comando INPUT, risultano trasparenti.

Segue il programma CASS3 che esemplifica la scrittura di campi numerici.

```
100 REM CASS3
110 REM CARICAMENTO FILE NUMERICO
120 REM OGNI RECORD LOGICO COMPRENDE 2 CAMPI
130 REM IL PRIMO E' UN NUMERO INTERO
140 REM IL SECONDO UN NUMERO DECIMALE
150 REM SI USA CHR$(13) COME SEPARATORE DI CAMPO
160 REM PER USCIRE DARE UN INTERO NULLO
170 REM IL RETURN FINALE LO METTE IL SISTEMA
180 CH$=CHR$(13)
190 OPEN1,1,1,"NUMERICO"
193 PRINT"♥NUMERI INTERI <= 32767QQ"
195 PRINT"PER USCIRE RISPONDERE 0 (ZERO) A INTEROQQ"
200 INPUT"INTERO: ";I%
210 IFI%=0THENCLOSE1:STOP
220 INPUT"DECIMALE: ";D
230 PRINT#1,I%;CH$;D
240 GOTO200
```

Il programma scrive record composti da 2 campi numerici, il primo intero e il secondo decimale. Non si devono usare interi superiori a 32767.
Si provi il programma rispondendo come qui sotto indicato:

INTERO? 765
DECIMALE? 9.765
INTERO? 345
DECIMALE? .6543
INTERO? 234
DECIMALE? .78
INTERO? -456
DECIMALE? 9.456
INTERO? 0

Viene così creato un file di nome NUMERICO contenente 4 record di 2 campi ciascuno.

Si può ora provare il programma CASS4 che segue, il quale legge il file NUMERICO e lo stampa sulla stampante; quando ha finito di leggere, stampa anche il contenuto del buffer byte dopo byte. Si può così vedere come vengono trattati i campi numerici.

```
100 REM CASS4
110 REM LETTURA FILE NUMERICO
120 REM CON INPUT VENGONO LETTI I 2 CAMPI
130 REM ESSI VENGONO STAMPATI
140 REM QUANDO E' FINITO IL FILE VIENE LETTO
150 REM E STAMPATO IL CONTENUTO DEL BUFFER
160 OPEN1,1,0,"NUMERICO"
170 PRINT"]ACCENDI LA STAMPANTE"
180 GETA$:IFA$=""THEN180
190 OPEN4,4:PRINT#4,"CONTENUTO FILE NUMERICO":PRINT#4:PRINT#4
200 INPUT#1,I%,D:TS=ST
210 PRINT#4,I%,D
220 IFTS<>64THEN200
230 PRINT#4:PRINT#4
240 PRINT#4,"CONTENUTO BUFFER CASSETTA":PRINT#4:PRINT#4
250 I=828
260 FORK=1TO48:FORL=1TO4
270 PRINT#4,I;"  ";MID$(STR$(PEEK(I))+"  ",2,3);"  ";
280 I=I+1:NEXTL:PRINT#4
300 NEXTK:PRINT#4:PRINT#4:CLOSE1:CLOSE4:STOP
```

Risultati del programma CASS4:

```
CONTENUTO FILE NUMERICO

 765           9.765
 345           .6543
 234           .78
-456           9.456
```

La funzione MID$ alla linea 270 serve per incolonnare i dati in stampa. Si osservi la stampa del buffer. Si vede un 2 iniziale, che significa file di dati. Dal byte 829 al byte 833 c'è il numero 765 preceduto e seguito da uno spazio; infatti quando si chiede di scrivere un numero, il sistema esegue la funzione STR$ del numero, aggiungendo uno spazio o il segno prima e uno spazio dopo. Nel byte 834 si ha il 13 del RETURN, e dal byte 835 al byte 841 si ha il numero 9.765 con il punto (46) nel byte 837. Si osservi lo zero binario di fine file al byte 883.

Si noti che se nel programma CASS4 si modificano le linee 200 e 210 così:

```
200 INPUT#1,A$,B$
210 PRINT#4,A$,B$
```

il programma funziona ugualmente, è cioè indifferente leggere un campo, scritto come variabile numerica, come stringa o come numero.

Si può fare un'altra prova; porre nel programma CASS4 uno STOP dopo la OPEN di NUMERICO: 165 STOP e dare il RUN. Allo STOP scrivere in immediato: OPEN4,4 per attivare la stampante e di nuovo in immediato: GOTO250. Si ottiene sulla stampante il contenuto del buffer della cassetta dopo la OPEN e si vede il nome NUMERICO dal byte 833 al byte 840.

Risultati del programma CASS4 modificato:

CONTENUTO BUFFER CASSETTA

828	2	829	32	830	55	831	54
832	53	833	32	834	13	835	32
836	57	837	46	838	55	839	54
840	53	841	32	842	13	843	32
844	51	845	52	846	53	847	32
848	13	849	32	850	46	851	54
852	53	853	52	854	51	855	32
856	13	857	32	858	50	859	51
860	52	861	32	862	13	863	32
864	46	865	55	866	56	867	32
868	13	869	45	870	52	871	53
872	54	873	32	874	13	875	32
876	57	877	46	878	52	879	53
880	54	881	32	882	13	883	0
884	32	885	32	886	32	887	32
888	32	889	32	890	32	891	32
892	32	893	32	894	32	895	32
896	32	897	32	898	32	899	32
900	32	901	32	902	32	903	32
904	32	905	32	906	32	907	32
908	32	909	32	910	32	911	32
912	32	913	32	914	32	915	32
916	32	917	32	918	32	919	32
920	32	921	32	922	32	923	32

Per completare l'argomento si suggerisce di provare il programma CASS5 che segue. In esso viene scritto un file contenente 13 record di 26 caratteri ciascuno, proprio per fargli occupare 2 blocchi fisici. Vengono stampati i contenuti del buffer in diversi momenti: dopo l'apertura del file, dopo la scrittura di un solo record, prima di chiudere il file (non si ha ancora lo zero binario nel byte 989 e si vede che il buffer non viene pulito tra una scrittura e l'altra), dopo la chiusura con carattere di fine file,dopo la lettura del primo blocco e dopo la lettura del secondo (questa volta si vede lo zero binario di chiusura). Il file viene riletto usando la GET.

```
100 REM CASS5
110 REM SCRITTURA DI UN FILE SU CASSETTA COMPOSTO
120 REM DA 13 RECORD DI 26 CARATTERI CIASCUNO
130 REM NUMERO TOTALE DI CARATTERI:(26+1)*13=351
140 REM OGNI RECORD 26 CAR. + CHR$(13)
150 REM I RECORD SONO AMMUCCHIATI NEL BUFFER PER 191
160 REM CAR. E L'ULTIMO RECORD VIENE SPEZZZATO
170 REM CONTINUANDOLO NEL BLOCCO FISICO SEGUENTE
160 REM OCCORRONO 2 BLOCCHI FISICI
190 REM NEL PRIMO 191 CAR. E NEL SECONDO 160
200 PRINT"]ACCENDI LA STAMPANTE"
210 GETB$:IFB$=""THEN210
220 OPEN4,4:A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
230 REM SI APRE IL FILE PER SCRIVERE CON SEGNALAZIONE DI FINE NASTRO
240 OPEN1,1,2,"PROVA BUFFER"
250 PRINT#4,"CONTENUTO BUFFER DOPO OPEN":PRINT#4
255 GOSUB1000
260 PRINT#1,A$:REM SCRITTURA PRIMO RECORD
270 REM STAMPA BUFFER DOPO SCRITTURA PRIMO RECORD
275 PRINT#4:PRINT#4
280 PRINT#4,"CONTENUTO BUFFER DOPO PRIMO RECORD":PRINT#4
285 GOSUB1000
290 REM SCRITTURA ALTRI 12 RECORD
300 FORK=1TO12:PRINT#1,A$:NEXTK
310 REM STAMPA BUFFER CON SECONDO BLOCCO FISICO
315 PRINT#4:PRINT#4
320 PRINT#4,"CONTENUTO BUFFER ULTIMO BLOCCO":PRINT#4
325 GOSUB1000
330 CLOSE1:REM CHIUSURA FILE CASSETTA
340 REM STAMPA BUFFER DOPO CHIUSURA FILE
345 PRINT#4:PRINT#4
350 PRINT#4,"CONTENUTO BUFFER DOPO CHIUSURA FILE":PRINT#4
355 GOSUB1000
360 REM LETTURA E STAMPA FILE
370 PRINT"XXRIAVVOLGI CASSETTA"
380 GETB$:IFB$=""THEN380
390 PRINT#4,"13 RECORD LETTI CON INPUT":PRINT#4
400 OPEN1,1,0,"PROVA BUFFER"
410 INPUT#1,B$:TS=ST:PRINT#4,B$
420 IFTS=64THENCLOSE1:GOTO440
430 GOTO410
440 PRINT#4:PRINT#4
450 REM LETTURA CON GET E STAMPA FILE
460 PRINT"XXRIAVVOLGI CASSETTA"
470 GETB$:IFB$=""THEN470
475 PRINT#4:PRINT#4
480 PRINT#4,"CARATTERI LETTI CON GET":PRINT#4
```

```
490 OPEN1,1,0,"PROVA BUFFER"
500 GET#1,B$:TS=ST:PRINT#4,ASC(B$);
510 IFTS=64THENCLOSE1:GOTO550
520 GOTO500
530 REM CONTENUTO BUFFER DOPO TRASFERIMENTO PRIMO BLOCCO
540 REM FISICO PER EFFETTO PRIMO GET
550 PRINT#4:PRINT#4:PRINT"[illegible]RIAVVOLGI CASSETTA"
560 GETB$:IFB$=""THEN560
565 PRINT#4:PRINT#4
570 PRINT#4,"CONTENUTO BUFFER DOPO LETTURA PRIMO BLOCCO"
580 PRINT#4:OPEN1,1,0,"PROVA BUFFER"
590 GET#1,B$:GOSUB1000
600 REM LETTURA ALTRI 191 CARATTERI PER PASSARE
610 REM AL SECONDO BLOCCO
620 FORK=1TO191:GET#1,B$:NEXTK
625 PRINT#4:PRINT#4
630 PRINT#4,"CONTENUTO BUFFER DOPO LETTURA SECONDO BLOCCO"
640 PRINT#4:GOSUB1000:CLOSE1:CLOSE4:STOP
999 REM ROUTINE STAMPA BUFFER
1000 I=828
1010 FORK=1TO48:FORL=1TO4
1020 PRINT#4,I;" ";MID$(STR$(PEEK(I))+"   ",2,3);
1030 I=I+1:NEXTL:PRINT#4
1040 NEXTK:PRINT#4:PRINT#4:RETURN
```

Risultati del programma CASS5:

CONTENUTO BUFFER DOPO PRIMO RECORD

828	2	829	65	830	66	831	67
832	68	833	69	834	70	835	71
836	72	837	73	838	74	839	75
840	76	841	77	842	78	843	79
844	80	845	81	846	82	847	83
848	84	849	85	850	86	851	87
852	88	853	89	854	90	855	13
856	32	857	32	858	32	859	32
860	32	861	32	862	32	863	32
864	32	865	32	866	32	867	32
868	32	869	32	870	32	871	32
872	32	873	32	874	32	875	32
876	32	877	32	878	32	879	32
880	32	881	32	882	32	883	32
884	32	885	32	886	32	887	32
888	32	889	32	890	32	891	32
892	32	893	32	894	32	895	32
896	32	897	32	898	32	899	32
900	32	901	32	902	32	903	32
904	32	905	32	906	32	907	32
908	32	909	32	910	32	911	32
912	32	913	32	914	32	915	32
916	32	917	32	918	32	919	32
920	32	921	32	922	32	923	32
924	32	925	32	926	32	927	32
928	32	929	32	930	32	931	32
932	32	933	32	934	32	935	32
936	32	937	32	938	32	939	32
940	32						

CONTENUTO BUFFER DOPO OPEN

828	2	829	1	830	16	831	4
832	24	833	80	834	82	835	79
836	86	837	65	838	32	839	66
840	85	841	70	842	70	843	69
844	82	845	32	846	32	847	32
848	32	849	32	850	32	851	32
852	32	853	32	854	32	855	32
856	32	857	32	858	32	859	32
860	32	861	32	862	32	863	32
864	32	865	32	866	32	867	32
868	32	869	32	870	32	871	32
872	32	873	32	874	32	875	32
876	32	877	32	878	32	879	32
880	32	881	32	882	32	883	32
884	32	885	32	886	32	887	32
888	32	889	32	890	32	891	32
892	32	893	32	894	32	895	32
896	32	897	32	898	32	899	32
900	32	901	32	902	32	903	32
904	32	905	32	906	32	907	32
908	32	909	32	910	32	911	32
912	32	913	32	914	32	915	32
916	32	917	32	918	32	919	32
920	32	921	32	922	32	923	32
924	32	925	32	926	32	927	32
928	32	929	32	930	32	931	32
932	32	933	32	934	32	935	32
936	32	937	32	938	32	939	32
940							32

CARATTERI LETTI CON GET

```
65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
85 86 87 88 89 90 13 65 66 67 68 69 70 71 72 73 74 75 76 77
78 79 80 81 82 83 84 85 86 87 88 89 90 13 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
13 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83
84 85 86 87 88 89 90 13 65 66 67 68 69 70 71 72 73 74 75 76
77 78 79 80 81 82 83 84 85 86 87 88 89 90 13 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
90 13 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82
83 84 85 86 87 88 89 90 13 65 66 67 68 69 70 71 72 73 74 75
76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 13 65 66 67 68
69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88
89 90 13 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
82 83 84 85 86 87 88 89 90 13 65 66 67 68 69 70 71 72 73 74
75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 13 65 66 67
68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87
88 89 90 13 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90 13
```

CONTENUTO BUFFER DOPO LETTURA PRIMO BLOCCO

```
828 2   829 65  830 66  831 67       884 66  885 67  886 68  887 69
832 68  833 69  834 70  835 71       888 70  889 71  890 72  891 73
836 72  837 73  838 74  839 75       892 74  893 75  894 76  895 77
840 76  841 77  842 78  843 79       896 78  897 79  898 80  899 81
844 80  845 81  846 82  847 83       900 82  901 83  902 84  903 85
848 84  849 85  850 86  851 87       904 86  905 87  906 88  907 89
852 88  853 89  854 90  855 13       908 90  909 13  910 65  911 66
856 65  857 66  858 67  859 68       912 67  913 68  914 69  915 70
860 69  861 70  862 71  863 72       916 71  917 72  918 73  919 74
864 73  865 74  866 75  867 76       920 75  921 76  922 77  923 78
868 77  869 78  870 79  871 80       924 79  925 80  926 81  927 82
872 81  873 82  874 83  875 84       928 83  929 84  930 85  931 86
876 85  877 86  878 87  879 88       932 87  933 88  934 89  935 90
880 89  881 90  882 13  883 65       936 13  937 65  938 66  939 67
```

CONTENUTO BUFFER ULTIMO BLOCCO

```
828 2   829 67  830 68  831 69
832 70  833 71  834 72  835 73
836 74  837 75  838 76  839 77
840 78  841 79  842 80  843 81
844 82  845 83  846 84  847 85
848 86  849 87  850 88  851 89
852 90  853 13  854 65  855 66
856 67  857 68  858 69  859 70
860 71  861 72  862 73  863 74
864 75  865 76  866 77  867 78
868 79  869 80  870 81  871 82
872 83  873 84  874 85  875 86
876 87  877 88  878 89  879 90
880 13  881 65  882 66  883 67
884 68  885 69  886 70  887 71
888 72  889 73  890 74  891 75
892 76  893 77  894 78  895 79
896 80  897 81  898 82  899 83
900 84  901 85  902 86  903 87
```

13 RECORD LETTI CON INPUT

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

5.4 ESEMPIO FILE SEQUENZIALE

Si riporta un esempio di programma che crea e gestisce un file sequenziale su cassetta. Il programma, di nome GESFILE, tratta un file di indiri[illegible]ord logico comprende i seguenti campi:

.COGNOME
.NOME
.INDIRIZZO
.CITTA'
.CAP
.TELEFONO

tutti di lunghezza variabili e tutti trattati come stringhe. Si usa come separatore di campo il RETURN; quindi lo stesso separatore divide tra loro sia i campi che i record. Ogni campo non può superare i 79 caratteri; se più lungo viene troncato. L'archivio viene creato in ordine alfabetico per cognome e nome, e vengono rifiutati elementi fuori sequenza; questi potranno essere immessi in una successiva fase di aggiornamento. Non sono accettati cognomi e nomi uguali. Il numero di record trattabili dipende da due elementi: la lunghezza del nastro e la quantità di memoria disponibile. Infatti il programma funziona con qualunque configurazione di memoria del VIC20, ma se si dispone di poca memoria si possono trattare pochi record, altrimenti in fase di aggiornamento questi non entrano in memoria. Il programma esegue le seguenti operazioni:

.crea ex-novo il file,
.lista il file sulla stampante,
.aggiorna il file dopo averlo trasferito tutto in una matrice in memoria.

La fase di aggiornamento è la più delicata; si devono prima operare le variazioni dei record già esistenti (modifica di alcuni dati), poi la cancellazione dei record che non servono più, e infine l'aggiunta di eventuali nuovi record, che devono essere forniti in ordine alfabetico per cognome e nome. In fase di aggiunta record, il file viene riscritto su cassetta fino al record che precede quello da aggiungere, poi viene aggiunto il nuovo record sul nastro e viene chiesto se se ne vuole aggiungere un altro, continuando così fino alla fine.

```
1 REM GESFILE
4 PR=4:NF=4
7 CH$=CHR$(13):SP$="   "
10 PRINT"♥CREAZIONE FILE S/N":INPUT R$
13 IFR$<>"S"THEN46
16 GOSUB277:GOSUB286:GOSUB289:GOSUB292
19 LC$="":LN$="":K1=N:FORK=1TON
22 GOSUB184
25 IFSW<>0THENK1=K:GOTO43
28 IFC$>LC$THEN37
31 IFC$=LC$THENIFN$>LN$THEN37
34 GOSUB316:GOSUB317:GOTO22
37 LC$=C$:LN$=N$
```

```
40 GOSUB217:NEXTK
43 CLOSE1:PRINT"XFINITO CARICAMENTO   ";K1;" RECORD":STOP
46 PRINT"⊐STAMPA FILE S/N":INPUTR$
49 IFR$<>"S"THEN76
52 GOSUB295:GOSUB277:GOSUB286:GOSUB304:NR=4
55 PRINT#NF,"LISTA FILE ";NF$:PRINT#NF:PRINT#NF
58 GOSUB307
61 PRINT#NF,C$;SP$;N$
64 PRINT#NF,I$;SP$;CP$;SP$;L$;SP$;T$
67 PRINT#NF:PRINT#NF:NR=NR+4
70 IFFS=64THENCLOSE1:CLOSENF:PRINT"XXFINITO LISTA":STOP
73 IFNR=60THENFORK=1TO6:PRINT#NF:NEXTK:NR=0
75 GOTO58
76 PRINT"⊐AGGIORNAMENTO FILE"
79 PRINT"XXMONTA NASTRO VECCHIO"
82 GETA$:IFA$=""THEN82
85 GOSUB286:GOSUB289
88 DIMC1$(N),N1$(N),I1$(N),P1$(N),L1$(N),T1$(N)
91 GOSUB304
94 FORK=1TON
97 INPUT#1,C1$(K),N1$(K),I1$(K),P1$(K),L1$(K),T1$(K)
100 IFST=64THENCLOSE1:K1=K:K=N:NEXTK:GOTO106
103 NEXTK:PRINT"XFILE SUPERA NUM.MASS.REC.";N:STOP
106 PRINT"XLETTI ";K1;" RECORD":SW=0:N=K1
109 PRINT"XXVARIAZIONI S/N ":INPUTR$
112 IFR$<>"S"THEN121
115 GOSUB184:IFSW<>0THEN121
118 GOSUB220:GOTO115
121 PRINT"XCANCELLAZIONI S/N ":INPUTR$
124 IFR$<>"S"THEN139
127 PRINT"RISPONDI **** PER     USCIRE"
130 INPUT"COGNOME ";C$:IFC$="****"THEN139
133 INPUT"NOME ";N$:GOSUB235
136 GOTO130
139 SW=0:FF=0:K1=0:N1=1
142 PRINT"XPREPARA NUOVO NASTRO":GOSUB277
145 GOSUB286:GOSUB292:PRINT"XINSERIMENTI S/N ":INPUTR$
148 IFR$<>"S"THEN175
151 GOSUB184:IFSW<>0THEN172
154 IFFF=1THEN160
157 GOSUB247
160 IFC$>LC$GOTO169
163 IFC$=LC$ANDN$>LN$THEN169
166 GOSUB316:GOTO151
169 GOSUB217:LC$=C$:LN$=N$:K1=K1+1:GOTO151
172 IFFF=1THEN178
175 GOSUB268
178 PRINT"XFINITO AGGIORNAMENTO"
181 PRINT"SCRITTI ";K1;" RECORD":CLOSE1:STOP
184 PRINT"⊐PER USCIRE COGNOME =  ****XX"
187 SW=0:INPUT"COGNOME ";C$:IFC$="****"THENSW=1:RETURN
190 INPUT"NOME ";N$:INPUT"INDIRIZZO ";I$
193 INPUT"CAP ";CP$:INPUT"CITTA' ";L$:INPUT"TEL. ";T$
194 PRINT"XCONFERMI S/N":INPUTR$
195 IFR$="N"THEN184
196 IFLEN(N$)>79THENN$=LEFT$(N$,79)
199 IFLEN(C$)>79THENC$=LEFT$(C$,79)
202 IFLEN(I$)>79THENI$=LEFT$(I$,79)
205 IFLEN(CP$)>79THENCP$=LEFT$(CP$,79)
208 IFLEN(L$)>79THENL$=LEFT$(L$,79)
```

```
211 IFLEN(T$)>79THENT$=LEFT$(T$,79)
214 RETURN
217 PRINT#1,C$;CH$;N$;CH$;I$;CH$;L$;CH$;CP$;CH$;T$:RETURN
220 FORK=1TON:IFC1$(K)=C$ANDN1$(K)=N$THEN229
223 NEXTK
226 GOSUB241:GOTO232
229 I1$(K)=I$:LI$(K)=L$:P$(K)=CP$:T1$(K)=T$:K=N:NEXTK
232 RETURN
235 FORK=1TON:IFC1$(K)=C$ANDN1$(K)=N$THEN244
238 NEXTK
241 PRINT"XNON TROVATO NOME ";C$;SP$;N$:GOSUB317:RETURN
244 C1$(K)=SP$:K=N:NEXTK:RETURN
247 FORK=N1TON:IFC1$(K)=SP$THEN263
250 IFC1$(K)<C$THEN262
253 IFC1$(K)=C$ANDN1$(K)<N$THEN262
256 IFC1$(K)=C$ANDN1$(K)=N$THENPRINT"NOMI UGUALI":GOSUB317
259 N1=K:K=N:NEXTK:RETURN
262 GOSUB310:K1=K1+1:LC$=C1$(K):LN$=N1$(K)
263 NEXTK
265 FF=1:RETURN
268 IFN1=NANDFF=1THENRETURN
271 FORK=N1TON:IFC1$(K)=SP$THEN275
274 GOSUB310:K1=K1+1
275 NEXTK:RETURN
277 PRINT"XMONTA NASTRO":GOSUB317
283 RETURN
286 INPUT"XNOME FILE ";NF$:RETURN
289 INPUT"XQUANTI RECORD ";N:RETURN
292 OPEN1,1,2,NF$:RETURN
295 PRINT"XACCENDI LA STAMPANTE":GOSUB317
301 OPENNF,PR:RETURN
304 OPEN1,1,0,NF$:RETURN
307 INPUT#1,C$,N$,I$,CP$,L$,T$:FS=ST:RETURN
310 PRINT#1,C1$(K);CH$;N1$(K);CH$;I1$(K);CH$;P1$(K);CH$;
313 PRINT#1,L1$(K);CH$;T1$(K):RETURN
316 PRINT"XNOMI FUORI ORDINE ";C$;SP$;N$:RETURN
317 GETA$:IFA$=""THEN317
318 RETURN
999 END
```

Risultati in stampa:

```
LISTA FILE PROVA


ALESSI   PINO
CORSO COMO 89   MILANO   43212   87654321


CARLI NINO   NINO
CORSO VENEZIA 23   COMO   54321   9876543
```

Il programma occupa 2704 byte, per cui se si usa un VIC senza espansione di memoria restano solo 879 byte per le variabili e si possono gestire solo file corti. In questo programma si usa la tecnica dei sottoprogrammi, che si trovano a partire dalla linea 184. La fase di creazione del file va dalla linea 10 alla linea 43; quella di lista dalla 46 alla 75. La fase di aggiornamento inizia alla linea 76; alla linea 88 vengono dimensionati 6 vettori per contenere i campi dei record del file da modificare in memoria. Da 91 a 106 viene letto in memoria il file vecchio e, in caso si incontrino più record del previsto, si ha uno STOP. Da 109 a 118 vengono fatte le variazioni dei record; da 121 a 139 le cancellazioni e da 145 a 181 gli eventuali inserimenti di nuovi record. La sequenza delle operazioni di aggiornamento è necessariamente rigida: modifiche, cancellazioni, aggiunte. La ricerca dei record viene fatta in base al cognome e nome. Si fa notare che nelle linee da 196 a 211 vengono troncati i campi se hanno più di 79 caratteri.
A questo programma si possono fare alcune critiche; la principale è che, in fase di ingresso dati, quando viene chiesto di confermare l'esattezza di quanto si è scritto, se la risposta è "no", si deve riscrivere tutto il record.
Segue un pezzo di programma per gestire l'ingresso dati dove, invece, i campi di input sono numerati da 1 a 6, e alla risposta "no" viene chiesto quale campo si vuole variare e il nuovo dato, senza dover riscrivere tutto.

```
100 REM GESVIDEO
110 DIMD$(6),I$(6)
120 D$(1)="COGNOME":D$(2)="NOME":D$(3)="INDIRIZZO"
130 D$(4)="CAP":D$(5)="CITTA'":D$(6)="TEL."
140 SW=0:GOSUB300:IFSW=1THEN500
150 GOSUB350
160 GOTO500
300 REM INGRESSO DATI
310 PRINT"";:K=1:GOSUB400:IFI$(1)="****"THENSW=1:RETURN
320 FORK=2TO6:GOSUB400:NEXTK:RETURN
350 PRINT"CONFERMI S/N":INPUTR$
360 IFR$="S"THENRETURN
370 INPUT"QUALE CAMPO ";K
380 INPUT"CAMPO ";I$(K)
390 GOSUB450:GOTO350
400 PRINTK;" ";D$(K);" ";:INPUTI$(K):RETURN
450 PRINT"";:FORK=1TO6:PRINTK;" ";D$(K);" ";I$(K):NEXTK:RETURN
500 STOP
```

Come si vede dal listato, sono stati definiti dei vettori sia per le descrizioni dei dati di ingresso che per i dati stessi. In tale modo si possono usare dei cicli e il numero del campo da modificare serve come indice per muoversi nei vettori. La variabile SW serve per sapere se si sono caricati 4 asterischi per il cognome per uscire.

CAPITOLO 6

FILE SU DISCO

6.1 UNITA' 1540

L'unità 1540 è, a differenza dell'ünità a cassette, una periferica intelligente. Questo significa che in essa, oltre ai necessari meccanismi, è contenuto un processore che lavora, dopo aver ricevuto comandi dalla CPU, in modo indipendente. Le parti componenti l'unità sono: il processore 6502, 16K di memoria ROM contenenti il DOS (Disk Operating System) per la gestione delle operazioni disco, 2K di memoria RAM per i buffer (memorie di transito per lettura e scrittura) e le memorie di lavoro, il motore per la rotazione del dischettto e quello per il movimento della testina di lettura/scrittura, i meccanismi per il controllo delle operazioni, per il movimento della testina di lettura e scrittura e per il posizionamento sulle tracce. Le

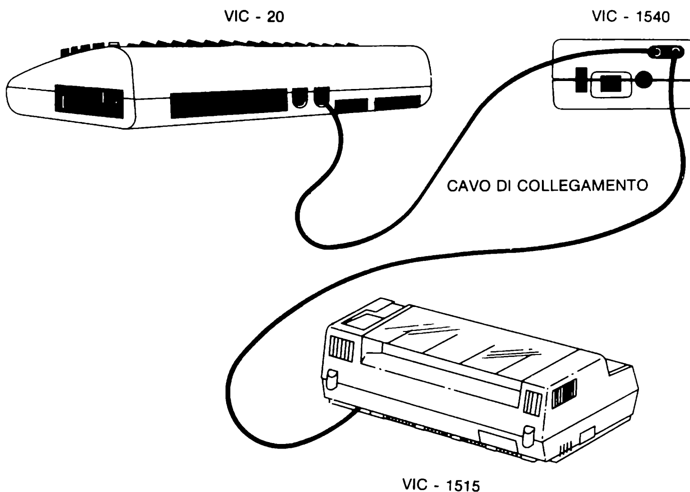


Figura 6.1 Collegamento tra calcolatore, unità floppy e stampante

operazioni disco non consumano spazio addizionale di memoria centrale, salvo quello necessario per la stesura delle relative frasi di programma. L'interfaccia di collegamento tra l'unità disco e il calcolatore è di tipo seriale IEEE 488, ed è la stessa usata per la stampante VIC-1515 (figura 6.1).
Nella parte anteriore dell'unità sono presenti due indicatori luminosi (figura 6.2): quello più esterno, verde, se acceso indica che l'unità è pronta per lavorare, quello più interno, rosso, resta acceso quando è in svolgimento una operazione disco, mentre lampeggia quando si verifica un errore.

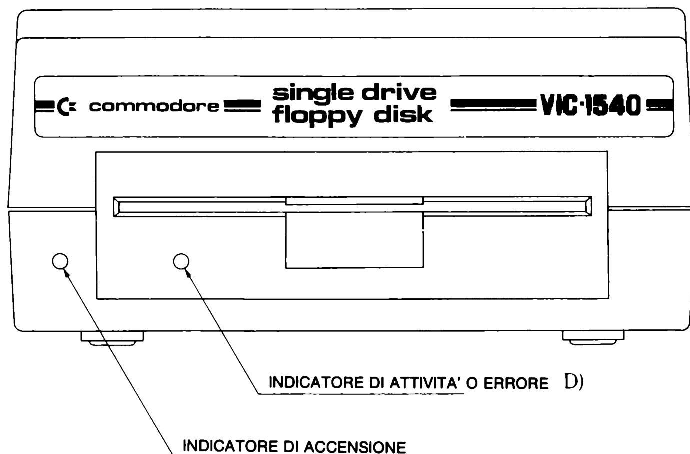


Figura 6.2 Parte anteriore unità 1540

L'utente deve prestare attenzione all'indicatore rosso; se esso pulsa, le operazioni disco in corso non vanno a buon fine ed è consigliabile interrompere il lavoro.
Si riassumono nella Tabella seguente le caratteristche principali dell'Unità 1540.

Capacità totale	174848 byte per disco (683x256)
Sequenziale	168656 byte per disco (664x256-2x664 byte di concatenazione)
Relativa	167132 byte per disco (664x256-2x664-6x254) 65535 massimo numero record logici per file
Capacità directory	144 file per disco
Settori per traccia	da 17 a 21
Byte per settore	256
Tracce	35
Blocchi	683 (664 blocchi liberi)

6502	processore
6522(2)	I/O
2114(4)	buffer 2K RAM
Altezza	97 mm
Larghezza	200 mm
Profondità	374 mm
Tensione	100,120,220 o 240 VAC
Frequenza	50 o 60 Hz
Potenza	25 Watt
Dischetti	standard mini 5 1/4", singola faccia singola densità

6.2 FLOPPY DISK

Il FLOPPY o DISCHETTO è un disco flessibile del diametro di 5 pollici e 1/4; esso è magnetizzabile ed è contenuto in una busta protettiva di plastica. Sulla busta sono visibili: una apertura oblunga che viene usata per la comunicazione tra il dischetto e la testina di lettura/scrittura, un foro circolare che serve per il corretto allineamento, e una apertura laterale rettangolare che serve per proteggere contro la scrittura, se si ricopre con una apposita etichetta. Si veda la figura 6.3, nella quale la freccia indica la direzione di inserzione nell'unità.

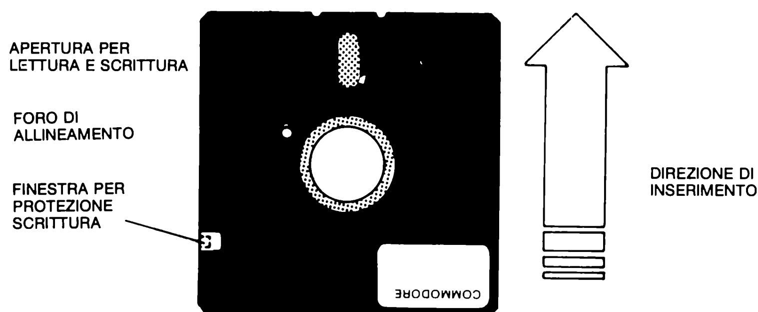


Figura 6.3 Floppy Disk

Il dischetto deve essere maneggiato con cura, toccando solo le parti protette dalla busta e con le seguenti precauzioni:

- non tenerlo troppo vicino al calcolatore o alle sue periferiche;
- non tenerlo vicino a motori elettrici o calamite;

.inserirlo nell'unità quando è già attiva (luce verde accesa);
.toglierlo dall'unità prima di disattivarla;
.non esporlo alla luce o al calore intensi;
.scrivere sulla etichetta con una penna morbida;
.non strofinare in alcun modo la sua superficie.

Il floppy è del tipo singola faccia, singola densità, a 40 tracce; cioè la sua superficie viene utilizzata da una sola parte, le registrazioni sono a singola densità e possono essere usate solo 40 tracce (cerchi concentrici) sulla superficie. In realtà il VIC utilizza solo 35 tracce per memorizzare. Nella figura 6.4 è riportato uno schema, non in scala, del dischetto. In essa vengono mostrate alcune tracce, il foro per l'allineamento (che esiste anche sul dischetto oltre che sulla busta), e un settore. Per settore si intende una parte di traccia, delimitata idealmente da 2 raggi che partono dal centro. Ogni traccia è divisa in un numero definito di settori, numero che varia passando da un gruppo di tracce all'altro.

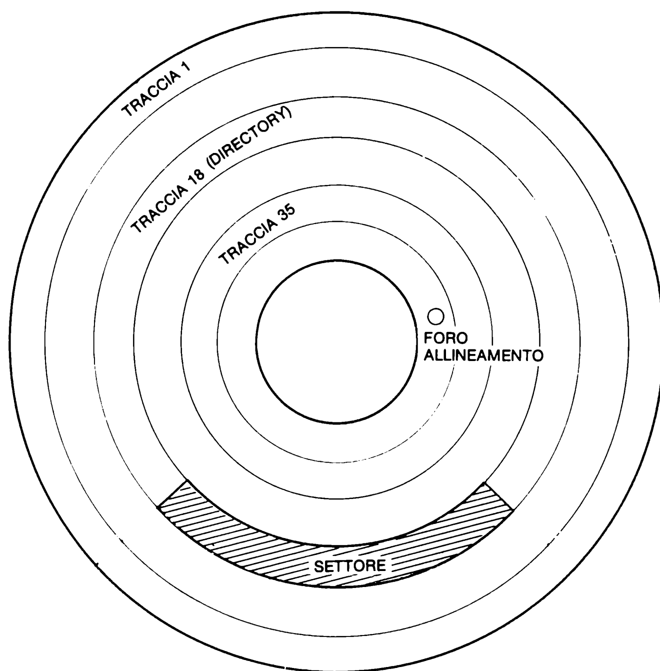


Figura 6.4 Schema di un floppy

Nella tabella che segue si riporta la distribuzione dei settori nelle 35 tracce disponibili:

Numero traccia	Numero settori	Numerazione settori
da 1 a 17	21	da 0 a 20
da 18 a 24	19	da 0 a 18
da 25 a 30	18	da 0 a 17
da 31 a 35	17	da 0 a 16

sono quindi disponibili: 17 tracce da 21 settori,
7 tracce da 19 settori
6 tracce da 18 settori
5 tracce da 17 settori
per un totale di 683 settori su 35 tracce.

Il dischetto non può essere usato se non contiene un indice di riferimento, come un qualunque libro. Per creare e mantenere questo indice, chiamato directory, il sistema usa la traccia 18, di 19 settori. All'utente restano disponibili 664 settori per il suo lavoro (683-19=664).
In ogni settore sono disponibili per l'utente 256 caratteri; quindi si può dire che la capacità di un dischetto è di 169.984 caratteri (664x256=169.984).

Quando si comincia a usare un dischetto nuovo è necessario registrare su di esso gli indirizzi delle tracce e dei settori, usando una particolare istruzione; dopo l'esecuzione di questa operazione, che viene chiamata FORMATTAZIONE, il dischetto è pronto per l'uso e su di esso sono registrati gli indirizzi di traccia e settore (in ogni settore) e il nome per identificarlo. Durante la formattazione, il foro di allineamento viene usato come riferimento di inizio traccia.

In realtà ogni settore del dischetto contiene più dei 256 caratteri dichiarati precedentemente; infatti in ogni settore vengono registrati anche alcuni caratteri di identificazione e di controllo, oltre agli indirizzi di traccia e settore. Tra l'altro viene registrato alla fine del settore un numero che rappresenta la somma di tutti i bit 1 registrati (checksum) nella parte disponibile all'utente e serve per controllare la bontà della registrazione (un controllo analogo è presente nella parte del settore utilizzata dal sistema). Si parla sempre di 256 caratteri, dato che questi sono quelli che l'utente può usare; tutti gli altri caratteri sono trasparenti per l'utente e li usa solo il sistema. Si ha poi una ulteriore diminuzione nei caratteri disponibili, dato che i primi 2 caratteri di ogni settore (dei 256) sono usati dal sistema per concatenare tra loro i settori e creare i file; possiamo dire che l'utente può usare 254 caratteri per settore. I byte di ogni settore, disponibili, si numerano partendo da 0; quindi il byte 0 e il byte 1 sono usati per concatenare i settori, quelli da 2 a 255 per i dati.

I settori sono separati uno dall'altro da una zona disco non utilizzata, chiamata GAP. Si usano indifferentemente le parole blocco o settore per indicare le zone utilizzabili nel dischetto.

La directory (traccia 18) si compone di 3 parti:
.tabella della disponibilità dei settori;
.nome del disco e identificazione del sistema;
.indice del disco.

La tabella di disponibilità dei settori si chiama BAM (Block Availability Map) ed è situata nel settore 0 della traccia 18 dove occupa 144 byte, da 0 a 143. Essa contiene:

.byte 0 e 1	18 e 01, indirizzi del primo settore dell'indice;
.byte 2	65, codice ASCII della lettera A che indica il formato dell'unità 1540;
.byte 3	non usato;
.byte 4/143	mappa dei settori occupati, un bit per settore (1=libero, 0=occupato).

La corretta gestione della BAM è assolutamente necessaria per poter lavorare con i dischetti; essa infatti consente di utilizzare i settori realmente disponibili e non quelli già usati.

Nei byte da 144 a 255 del settore 0 della traccia 18 sono contenuti:

.byte 144/161	nome del disco, max 18 caratteri, eventualmente con aggiunta di "spazi inversi";
.byte 162/163	identificazione disco, 2 caratteri;
.byte 164	160, codice ASCII dello spazio inverso;
.byte 165/166	50 e 65, codici ASCII di 2A, a indicare la versione del DOS;
.byte 167/170	160, come byte 164;
.byte 171/255	non usati.

I settori da 1 a 18 della traccia 18 sono usati per l'indice dei file registrati sul dischetto. Per ogni file sono usati 30 caratteri; ognuna di queste registrazioni viene chiamata "entrata" (ENTRY). La struttura dei settori è la seguente:

.byte 0 e 1	concatenamento al settore seguente (zero binario nel byte 0 se ultimo della catena);
.byte 2/31	entrata 1;
.byte 32/33	separatore;
.byte 34/63	entrata 2;

....

....

.byte 224/225 separatore;
.byte 226/255 entrata 8.

In ogni settore possono essere registrate 8 entrate e quindi in totale sono registrabili 144 entrate, cioè 144 file per dischetto (file contraddistinti da un nome).

Per ogni entrata sono registrati i seguenti elementi:

.byte 0	tipo = 0 cancellato (DEL)	
	128 + tipo =	1 sequenziale (SEQ)
		2 programma (PROG)
		3 utente (USER)
		4 relativo (REL)
.byte 1/2	traccia e settore del primo blocco del file	
.byte 3/18	nome file completato con spazi inversi (16 car.)	
.byte 19/25	non usati	
b.yte 26/27	traccia e settore del file memorizzato dopo una operazione OPEN@ (cioè rimemorizzazione di un file già esistente)	
.byte 28/29	numero blocchi file: byte basso seguito da byte alto.	

6.3 DOS

Il DOS (Disk Operating System) risiede nella ROM dell' unità disco e provvede alla esecuzione delle operazioni disco che vengono lanciate dal Sistema Operativo del VIC. I compiti del DOS sono essenzialmente quelli di eseguire le operazioni di Input/Output (cioè i trasferimenti di byte tra memoria centrale e disco, nelle 2 direzioni) insieme con il Sistema Operativo del VIC, di gestire la directory del disco nelle sue 3 parti, di cancellare i file e di copiarli. Il DOS gestisce quindi la comunicazione tra il disco e l'interfaccia seriale che lo collega al calcolatore.

Il comando OPEN stabilisce una comunicazione tra unità disco e calcolatore; il comando CLOSE la chiude. Il formato di questi due comandi è:

OPEN lfn, dn, sa

CLOSE lfn

dove

- lfn è il numero logico del file (da 0 a 255), che deve comparire nei successivi comandi relativi a operazioni su quel file;

- dn è il numero del dispositivo, determinato via hardware; per l'unità disco è 8;
- sa è l'indirizzo secondario, che, a seconda del valore assegnatogli, consente l'invio di comandi al DOS, oppure il trasferimento vero e proprio di dati tra disco e memoria centrale. Per uniformarsi alla terminologia adottata dalla Commodore, d'ora in poi "sa" sarà anche chiamato canale. I valori che può assumere variano da 0 a 15:

sa = 15 apre un canale per l'invio di comandi al DOS, e per segnalazione di errori da parte del DOS;

sa = 0-14 apre un canale per il trasferimento di dati, e precisamente

sa = 0 e sa = 1

sono utilizzati dal sistema come canali di trasferimento di file di programmi, con i comandi SAVE e LOAD, come si vedrà in seguito;

sa = 2-14

sono canali disponibili all'utente per qualsiasi operazione di trasferimento dati.

Si vedrà più avanti che nel comando OPEN può essere presente un altro parametro dopo "sa"; è una stringa che rappresenta il nome di un file, il tipo del file e il tipo di operazione che su quel file si vuole eseguire.

Quando il DOS riceve un comando di OPEN, esso assegna al canale richiesto lo spazio necessario per lavorare e mette a disposizione dell'operazione disco uno dei suoi buffer. Se l'operazione di OPEN è richiesta quando non è disponibile lo spazio necessario di memoria RAM nell'unità a disco, si ha una segnalazione di errore del tipo NO CHANNEL.

I buffer disponibili sono 8; di essi 4 sono usati per il canale dei comandi, le variabili di lavoro, il controllo delle richieste di operazioni disco e la BAM. In conseguenza restano solo 4 buffer disponibili e quindi si possono trattare contemporaneamente solo 4 o 3 file, a seconda che siano di tipo diretto o sequenziale.

Il DOS dell'unità 1540 in collegamento con il VIC 20 può gestire 4 tipi di file su disco. Essi sono i 2 tipi file già visti per la cassetta, e cioè i file di PROGRAMMI e i file SEQUENZIALI e, inoltre, i file RANDOM (collegabili con gli USER che sono sequenziali) e i file RELATIVI.

6.4 COMANDI PER LA GESTIONE DEL DISCO

In questo paragrafo si analizzano i comandi, che possono essere usati sia in modo immediato che da programma, e consentono di operare con i dischi. Nei prossimi paragrafi si tratteranno dettagliatamente le operazioni inerenti ai diversi tipi di file.

Prima di tutto si deve stabilire una comunicazione con il disco, con il comando

```
OPEN lfn,8,15
```

Di solito, forse per abitudine, o per ricordare un solo numero, si vede scritto il

comando come:

OPEN 15,8,15

esprimendo cioè il numero logico del file e il canale con lo stesso numero. Ovviamente quello che deve essere 15 è solo l'indirizzo secondario. "lfn" deve essere usato nelle operazioni PRINT# che vengono usate dopo la OPEN.
Queste operazioni hanno il formato

PRINT# lfn, stringa

e trasmettono al DOS, come stringa, i comandi disco da eseguire. Quelli disponibili in questo primo gruppo sono i seguenti:

NEW	formattazione dischetto
INITIALIZE	preparazione disco per l'uso
LOAD "$"	lettura directory in memoria
VALIDATE	sistemazione della BAM
COPY	copia di un file
RENAME	cambio nome a un file
SCRATCH	cancellazione di un file.

I comandi possono essere usati sia scrivendo il nome per esteso, che usando solo la lettera iniziale.

Il comando NEW serve per preparare un disco nuovo per l'uso o per rendere riutilizzabile ex-novo un disco che non serve più. Con questo comando vengono scritti sul disco gli indirizzi di traccia e settore, viene preparata la BAM, viene scritto il nome e la identificazione del disco e viene predisposto lo spazio per l'indice. Il nome del disco può essere lungo fino a 18 caratteri; l'identificazione è formata da 2 caratteri e serve a distinguere tra loro dischi con lo stesso nome. Il comando si scrive:

PRINT#lfn,"dr:fn,xx"

dove:

dr=numero unità; per il VIC 20 è sempre 0 e può essere omessa;
fn=nome-disco;
xx=identificazione disco (2 caratteri)

Esempi:

```
OPEN 15,8,15
PRINT#15,"N0:ANAGRAFICO,55"
```

apre il canale comandi, formatta un dischetto assegnandogli il nome "ANAGRAFICO" e la identificazione "55".

Lo stesso effetto si ottiene scrivendo il secondo comando così:

```
PRINT#15,"N:ANAGRAFICO,55"
```

Il comando può essere dato più sinteticamente anche così:

```
OPEN 15,8,15,"N:ANAGRAFICO,55"
```

Qualora nel comando si ometta la identificazione del disco, e si usi un disco già formattato, si ottiene di azzerare la BAM e di cancellare tutti i file presenti senza cambiare la identificazione, ma , volendo solo il nome. In tale caso l'operazione risulta più veloce.
Dopo questa operazione un dischetto è pronto per l'uso e contiene 0 file. E' buona norma applicare all'esterno del disco una etichetta recante il nome e la identificazione assegnati.

Il comando INITIALIZE serve per far caricare nella memoria dell'unità la BAM del dischetto, e per allineare la testina di lettura/scrittura all'inizio della traccia. L'operazione di caricamento della BAM in memoria è fondamentale per la buona riuscita delle operazioni disco; se non viene fatta si rischia di danneggiare in modo irreparabile il dischetto andando a scrivere su settori già occupati. Si raccomanda di INIZIALIZZARE IL DISCHETTO OGNI VOLTA CHE NE VIENE INSERITO UNO NUOVO NELL'UNITA'. Meglio farlo spesso che troppo di rado
Esempio:

```
OPEN 15,8,15
PRINT#15,"I0"                oppure
```

```
PRINT#15,"I"
```

oppure, più sinteticamente:

```
OPEN 15,8,15,"I"
```

Si noti che il comando viene accettato anche se non si scrivono gli apici di chiusura della stringa comando.

Il comando LOAD"$" serve per caricare nella memoria del calcolatore la directory del dischetto. Il precedente contenuto della memoria viene perso, quindi si deve memorizzare il programma eventualmente presente in memoria prima di usare il comando.
Scrivendo:

```
LOAD"$",8
LIST
```

si ottiene sul video la lista della directory (seconda e terza parte). La prima riga riporta il contenuto della seconda parte del settore 0 traccia 18, cioè il nome, la identificazione e la versione del DOS. Segue una riga per ogni file presente, con il numero dei blocchi usati, il nome e il tipo di file; l'ultima riga reca l'indicazione dei blocchi di disco ancora liberi.
La directory può essere listata sulla stampante come un qualunque file di programmi presente in memoria.

Il comando VALIDATE serve per rimettere ordine in un disco; si deve usare se è accaduto qualcosa di irregolare. Il comando controlla il disco e rigenera la BAM, considerando occupati i settori coinvolti nei file della directory; sono quindi esclusi i file RANDOM non riportati nella directory, che vengono in conseguenza distrutti, nel senso che i loro settori sono considerati liberi. Non si deve mai eseguire questo comando su dischetti che contengono file RANDOM (si veda il paragrafo relativo). Può capitare di vedere nella lista della directory dei file contrassegnati da un asterisco (sono file non correttamente chiusi); essi con il comando VALIDATE vengono cancellati. Si ricordi che un disco è in buone condizioni se la somma dei blocchi liberi e di quelli occupati dai file registrati nella directory, più eventuali blocchi usati per i file RANDOM, dà il numero 664.
Il comando si scrive:

```
OPEN 15,8,15
PRINT#15,"V0"
```

lo zero può essere omesso; oppure:

```
OPEN 15,8,15,"V"
```

Il comando COPY consente di ottenere copie dello stesso file con nomi diversi, oppure di fondere, concatenandoli, più file (fino a 4) in un unico file. Nel primo caso il comando si scrive:

```
OPEN 15,8,15
PRINT#15,"C:dfn=sfn"
```

dove "dfn" è il nome del file destinazione e "sfn" è il nome del file sorgente. I nomi dei file possono essere al massimo di 16 caratteri alfanumerici; se sono più corti, il sistema li completa con caratteri il cui codice corrisponde alla pressione contemporanea dei due tasti SHIFT e SPAZIO. Inoltre il sistema completa il nome del file con un'indicazione di tipo (primo byte della registrazione nella directory) che può variare da 0 a 4 (si veda la descrizione della directory nel paragrafo 6.2). Ai nomi dei file può essere premesso "0", prima dei due punti, come numero dell'unità.
Nel secondo caso si scrive:

```
PRINT#15,"C:dfn=sfn1,sfn2,sfn3,sfn4"
```

dove "dfn" è il nome del file destinazione e "sfni" sono i nomi dei file sorgenti da fondere (da 2 a 4). Si deve fare attenzione al fatto che la stringa dei comandi tra gli apici può essere lunga al massimo 40 caratteri. I file da copiare devono essere stati precedentemente chiusi.

Il comando RENAME serve per cambiare nome a un file; il nuovo nome scelto non deve essere già presente nel disco. Si scrive:

```
OPEN 15,8,15
PRINT#15,"R:nfn=ofn"
```

dove "nfn" è il nome nuovo che si vuole usare e "ofn" è il nome vecchio. Si noti che questa operazione non fa materialmente riscrivere un file spostandolo in un'altra zona del disco, ma modifica semplicemente il nome riportato nella directory. Come al solito si può usare "0" prima dei due punti per numero di unità. Il file vecchio deve essere stato correttamente chiuso.

Il comando SCRATCH serve per cancellare dal disco i file che non servono più, rendendo disponibile nella BAM lo spazio da essi occupato. Si scrive:

```
OPEN 15,8,15
PRINT#15,"S:fn1,fn2,..fnn"
```

dove "fni" sono i nomi dei file da cancellare (ovviamente vale anche per un solo file). Per effetto di questo comando, nella directory, al posto del file cancellato, viene scritto 0 nel tipo, che significa appunto "cancellato". Nella lista della directory, i file cancellati non compaiono.

Qualora si vogliano cancellare più file con nomi in parte simili si può usare:

* per rappresentare qualunque carattere (tutti o da un certo punto in poi)

? per indicare qualunque carattere in quella posizione.
esempio:

PIP* si riferisce a: PIPPO
PIPPOLO
PIPA

e simili

AA??BB si riferisce a:AAXXBB
AAZZBB

e simili.

Il comando PRINT#,"S:????.SEQ"

cancella tutti i file di tipo SEQ con nome di 4 caratteri.

Negli esempi precedenti, si è sempre scritto il comando OPEN prima di ogni altro comando; naturalmente la OPEN per il canale comandi va data una volta per tutte. Se la si ripete, prima di averlo chiuso, si ottiene un messaggio di errore. In tale caso si deve scrivere CLOSE 15 e poi aprire nuovamente il canale.

6.5 FILE DI PROGRAMMI

I file di programmi (PRG) sono trattati come per le cassette; sono disponibili i comandi SAVE, LOAD e VERIFY. Per queste operazioni non si deve preliminarmente dare il comando OPEN, che viene sottinteso dal sistema, con sa 0 o 1. In ogni caso, però, il disco deve essere inizializzato, con una OPEN sul canale 15, in modo che la BAM e la directory siano caricate nella RAM dell'unità disco. In questo caso si devono scrivere citando il numero dell'unità disco che è 8.

SAVE "0:fn",8

dove 0 può essere omesso e rappresenta il numero dell'unità disco, "fn" il nome del file e 8 è il numero dell'unità periferica. Con questa operazione il file viene memorizzato su disco e nella directory viene registrato il suo nome seguito dalla specifica PRG.
Il comando può essere scritto in modo leggermente diverso qualora si desideri memorizzare un programma con un nome già presente sul disco:

SAVE "@0:fn",8

dove il carattere "@" fa sì che il nuovo file programma vada a sostituire il vecchio con lo stesso nome. E' comunque consigliabile non usare spesso il comando SAVE in questo formato; infatti si può danneggiare la directory del disco. E' consigliabile

cancellare il vecchio file con il comando SCRATCH visto prima, eseguire la VALIDATE del disco (se non contiene anche file RANDOM) e poi salvare il nuovo programma senza usare il carattere "@".
Dopo il comando SAVE è consigliabile usare sempre il comando VERIFY per verificare che il programma memorizzato su disco sia identico a quello presente in memoria. Il comando si scrive:

```
VERIFY "0:fn",8          oppure
VERIFY "*",8             oppure
VERIFY "",8
```

si ottiene di confrontare il programma "fn", oppure l'ultimo scritto su disco, con quello presente in memoria.

Per caricare in memoria un programma memorizzato su dischetto si usa il comando:

```
LOAD "0:fn",8
```

dove 0 può essere omesso, "fn" è il nome del programma e 8 il numero dell'unità disco. Se non esiste su disco un programma con il nome "fn", la luce rossa di errore comincia a pulsare. Se si era aperto il canale 15 al momento dell'inizializzazione del disco, si deve dare il comando CLOSE 15 e rifare l'inizializzazione del disco, oppure scrivere PRINT#15,"I".

Quando il caricamento del programma avviene correttamente si vede apparire READY sul video e si può mandare in esecuzione il programma o con RUN o con GOTO n. Si deve però ricordare che l'operazione di caricamento chiude tutti i file aperti e quindi tutti i canali, compreso quello dei comandi (usato anche per i messaggi di errore). Se il programma lavora su file su disco, al suo interno devono essere contenuti tutti i comandi OPEN necessari.
Per leggere un file programma il sistema usa il canale 0, che non deve essere aperto dall'utente.

Anche nel caso del disco si può usare il comando SAVE nella forma:

```
SAVE "nome",8,1
```
per memorizzare il programma in modo tale che in fase di caricamento non venga rilocato.

Analogamente un comando di caricamento scritto:

```
LOAD "nome",8,1
```
carica un programma senza rilocarlo, anche se era stato salvato in modo normale.

Si rimanda alle osservazioni fatte al riguardo nel paragrafo 5.2, relativamente ai file di programmi su cassetta.

Per quanto riguarda la fusione di programmi, trattata sempre nel paragrafo 5.2, si consiglia di usare la cassetta come memoria di massa. Il lavoro potrebbe essere fatto anche usando il disco, ma si dovrebbe assegnare un nome diverso ad ogni spezzone di programma da memorizzare e sarebbe una cosa un pò noiosa.

6.6 FILE SEQUENZIALI

I file sequenziali su disco hanno caratteristiche analoghe a quelle dei file dello stesso tipo su cassetta. L'utente deve stabilire la struttura del record logico e dei campi che lo compongono; questi ultimi possono essere di lunghezza variabile e in conseguenza anche i record logici possono essere di lunghezza variabile. Non è necessario che l'utente si preoccupi del collegamento tra record logico e blocco fisico (settore disco), infatti la gestione del settore è a carico del sistema. I dati, sia numerici che di stringa, vengono trasferiti su disco carattere per carattere; ogni campo deve essere separato dal campo seguente da un separatore, che può essere il carattere virgola (CHR$(44)) o il carattere RETURN (CHR$(13)). Vale la regola che tra due RETURN non possono essere letti più di 87 caratteri. Il sistema usa un buffer di 256 caratteri (la dimensione di un settore disco in byte) per ammucchiare i dati da scrivere o per ottenere quelli da leggere. Il buffer viene messo a disposizione del file nel momento della sua apertura (OPEN), ma la posizione di memoria del buffer non è nota all'utente, come era, invece, per la cassetta.
La parola di stato ST può essere usata per ottenere la segnalazione di "fine file", rappresentata sempre dal numero 64; ST=0 significa che l'operazione ha dato buon esito. Qualora ST abbia un valore diverso si può analizzare attraverso il canale di errore (il 15) la situazione di 4 variabili: EN, EM$, ET, ES, per sapere cosa è successo. Il contenuto delle variabili è il seguente:

EN contiene il numero del messaggio di errore
EM$ contiene il messaggio
ET contiene la traccia disco che ha dato errore
ES contiene il settore disco che ha dato errore

Per leggerle occorre, dopo la OPEN sul canale 15, dare il comando

INPUT#15,EN,EM$,ET,ES

Il nome delle variabili può essere scelto a piacere; esse possono essere sia di stringa che numeriche, salvo la seconda che deve essere di stringa per la descrizione del messaggio di errore.
Segue la descrizione dei comandi disponibili.

OPEN lfn,dn,sa,"dn:fn,ft,modo"

dove i parametri hanno il significato seguente:

lfn	numero logico del file, usato in tutte le operazioni di lettura o scrittura sul file;
dn	8, numero della periferica disco;
sa	indirizzo secondario o canale (da 2 a 14);
dr	indirizzo dell'unità(0); si può omettere insieme ai ":" seguenti;
fn	nome del file da aprire; è il nome che compare nella directory;
ft	tipo del file, può essere: PRG (programmi), SEQ (sequenziali), USR (user) e REL (relativi)
modo	può essere R (READ) o W (WRITE) a seconda che si apra per leggere o scrivere.

Quando un file viene aperto per scrivere, si può far precedere al nome del file il carattere "@", prima dei ":", per specificare che si desidera cancellare un eventuale file con lo stesso nome già presente sul dischetto. I parametri: PRG, SEQ, USR, REL, READ e WRITE possono essere abbreviati usando solo la lettera iniziale. La stringa tra apici può essere sostituita tutta o in parte da variabili di stringa, concatenate con il "+".

Per chiudere un file, alla fine delle operazioni si usa il comando:

CLOSE lfn

dove lfn è il numero logico usato nella OPEN.

I file devono essere sempre chiusi quando il programma non li deve più usare; se il file è stato aperto per scrivere, l'operazione di chiusura scrive su disco l'ultimo blocco fisico, e in tutti i casi libera sia il numero logico assegnato al file che il canale, che possono essere usati nuovamente per altri file. Viene cioè aggiornata la tabella dei file nella RAM del VIC.

All'inizio di un programma che usa i file su disco è necessario aprire il canale 15 dei comandi, che viene usato anche per le segnalazioni di errore. Quando si chiude il canale 15 (CLOSE 15) vengono chiusi anche tutti gli altri canali, però per il programma i file sono ancora aperti da un punto di vista logico, cioè sono ancora presenti nella tabella dei file in RAM. In conseguenza è bene chiudere prima gli altri file con la CLOSE e poi il 15.
Qualora si abbia un messaggio di errore vengono chiusi tutti i file dal punto di vista logico per il programma, ma non vengono chiusi i canali per il disco; per poterli

chiudere correttamente si deve riaprire il canale comandi 15 e poi chiuderlo, così:

```
OPEN 15,8,15
CLOSE 15
```

Si ricordi che i comandi INITIALIZE, NEW e VALIDATE chiudono tutti i canali associati al disco e quindi rovinano i file che non sono stati prima chiusi correttamente.

Per scrivere dati nel file sequenziale si usa il comando PRINT#. Si è già visto l'uso di questo comando per inviare una stringa di comandi al disco. Il formato in questo caso è il seguente:

```
PRINT#lfn, dati
```

dove lfn è il numero logico del file usato nella OPEN (con il parametro W per scrivere), e "dati" è un insieme di dati da scrivere. Per i dati da scrivere valgono le stesse regole viste a proposito dei file sequenziali su cassetta; essi possono essere espressi come stringhe tra apici, numeri interi o decimali o nomi di variabili nei tipi consentiti. I dati vengono in ogni caso scritti su disco carattere per carattere, come se fossero delle stringhe; se sono numeri, il sistema prima di scriverli applica la funzione STR$. Si deve prestare molta attenzione ai separatori che si usano tra un dato e l'altro; la semplice virgola tra due nomi di variabili viene ignorata e le due variabili vengono conservate come una sola stringa. Per ottenere che venga conservata la virgola separatrice, essa deve essere forzata, cioè scritta tra apici (",") o come stringa con CHR$(44), e in questo caso i due dati potranno poi essere letti separatamente. Al posto della virgola separatrice si può usare anche CHR$(13), cioè il RETURN. Qualora la parte dati nel comando termini senza il carattere ";" il sistema aggiunge il carattere di fine record, che in questo caso viene influenzato dal valore del numero logico del file che si è usato. Per numero logico compreso tra 1 e 127 il sistema aggiunge il carattere corrispondente al RETURN, mentre per numero logico compreso tra 128 e 255 aggiunge 2 caratteri e precisamente il RETURN seguito dal carattere corrispondente a LINE FEED. In quest'ultimo caso questo secondo carattere andrebbe poi a costituire il primo carattere del campo successivo e potrebbe creare dei problemi. Per evitare complicazioni, si può prendere l'abitudine di mettere sempre come caratteri finali nella zona dei dati, CHR$(13) seguito da ";" Operando in questo modo il sistema non aggiunge alcun carattere e tutto funziona bene indipendentemente dal numero logico di file usato. Per esempio:

```
PRINT#lfn,a$CHR$(13)b$CHR$(13);
```

oppure:

```
PRINT#lfn,a$;CHR$(13);b$;CHR$(13);
```

sono istruzioni equivalenti, cioè producono la scrittura delle due stringhe separate dal RETURN e con un RETURN finale, qualunque sia il numero lfn.

Per leggere i dati si usa il comando:

INPUT#lfn,lista

dove lfn è il numero logico del file usato nella OPEN per leggere, e "lista" è una serie di nomi di variabili separate da virgola. In ogni variabile viene letto un campo dati formato da tanti caratteri quanti vengono trovati fino al primo separatore. I nomi delle variabili possono essere sia di tipo numerico che di stringa, ma se si tenta di leggere un campo alfanumerico in una variabile numerica si ha un errore. Ogni campo dati (o insieme di campi dati) compreso tra due RETURN non deve superare gli 87 caratteri (88 con il RETURN di destra).

Esiste un altra istruzione per leggere i dati carattere per carattere, senza escludere i caratteri separatori come succede con la precedente istruzione; essa è la GET#. Si scrive:

GET#lfn,nome variabile stringa

dove come al solito lfn è il numero logico del file usato nella OPEN per leggere, e il nome della variabile deve essere di stringa per poter leggere qualunque carattere. Con la GET non si ha limitazione sul numero di caratteri da leggere. I caratteri possono essere letti uno dopo l'altro e ammucchiati in una stringa con l'operazione di somma; così:

```
100 LET s$=""
110 FOR k=1 TO 100
120 GET#3,a$
130 LET s$=s$+a$
140 NEXT k
```

infatti la linea 100 pone s$ uguale alla stringa nulla e poi con il ciclo di GET vengono letti 100 caratteri e sistemati uno dopo l'altro nella strina s$.

Segue il programma APRISEQUENZIALI che dimostra quanti file sequenziali possono essere aperti contemporaneamente sul disco. Si vede che il numero massimo è 3; tentando di aprirne di più si ha un messaggio di errore.

```
1 REM APRISEQUENZIALI
5 OPEN15,8,15,"I"
7 INPUT"QUANTI FILE SEQ.";N
15 ONNGOTO60,50,40,30,20
20 OPEN6,8,6,"00:SEQ5,S,W":GOSUB100
```

```
21 PRINT#6,"RECORDSEQ5":GOSUB100
30 OPEN5,8,5,"@0:SEQ4,S,W":GOSUB100
31 PRINT#5,"RECORDSEQ4":GOSUB100
40 OPEN4,8,4,"@0:SEQ3,S,W":GOSUB100
41 PRINT#4,"RECORDSEQ3":GOSUB100
50 OPEN3,8,3,"@0:SEQ2,S,W":GOSUB100
51 PRINT#3,"RECORDSEQ2":GOSUB100
60 OPEN2,8,2,"@0:SEQ1,S,W":GOSUB100
61 PRINT#2,"RECORDSEQ1":GOSUB100
65 ONNGOTO74,73,72,71,70
70 CLOSE6:GOSUB100
71 CLOSE5:GOSUB100
72 CLOSE4:GOSUB100
73 CLOSE3:GOSUB100
74 CLOSE2:GOSUB100
99 CLOSE15:STOP
100 INPUT#15,EN,EM$,ET,ES
110 PRINTEN;EM$;ET;ES
120 RETURN
```

6.6.1 ESEMPIO ARCHIVIO SEQUENZIALE

Segue un programma per creare e gestire un file sequenziale su disco.

```
1 REM SEQDISCO
20 NF=4:PR=4:OPEN15,8,15:GOSUB3000
40 CH$=CHR$(13):SP$="     ":DIMI$(6),I1$(6)
43 DIMD$(6):D$(1)="COGNOME ":D$(2)="NOME ":D$(3)="TELEFONO "
44 D$(4)="INDIRIZZO ":D$(5)="CAP ":D$(6)="CITTA' "
55 PRINT"ℑCREAZIONE FILE  S/N ";:INPUTR$
60 IFR$<>"S"THEN500
65 GOSUB1270:GOSUB1250:GOSUB1280
80 PRINT#15,"I0"
85 OPEN2,8,2,"0:"+NF$+",S,W":GOSUB3000
90 LC$="":LN$=""
95 GOSUB1000
100 IFSW<>0THENCLOSE2:CLOSE15:PRINT"▓FINITO CARICAMENTO    FILE":STOP
110 IFI$(1)>LC$THEN160
150 IFI$(1)=LC$THENIFI$(2)>LN$THEN160
155 GOSUB1260:GOTO95
160 LC$=I$(1):LN$=I$(2)
165 GOSUB1200:GOTO95
500 PRINT"ℑSTAMPA FILE S/N";:INPUTR$
520 IFR$<>"S"THEN700
521 GOSUB1270:GOSUB1250:PRINT#15,"I0"
523 GOSUB1280:GOSUB1290
550 OPENNF,PR:PRINT#NF,"LISTA ARCHIVIO ";NF$:PRINT#NF
570 GOSUB1100:PRINT#NF,I$(1);SP$;I$(2)
575 PRINT#NF,I$(3):PRINT#NF,I$(4):PRINT#NF,I$(5);SP$;I$(6):PRINT#NF
600 IFFS=64THENCLOSE3:CLOSENF:CLOSE15:PRINT"▓FINITO LISTA":STOP
610 GOTO570
700 FF=0:PRINT"▓AGGIORNAMENTO FILE":GOSUB1270:GOSUB1250:PRINT#15,"I0"
710 GOSUB1280:GOSUB1290:OPEN2,8,2,"0:TEMP,S,W":GOSUB3000:LC$="":LN$=""
749 PRINT"ℑTIPO DI VARIAZIONE:"
750 PRINT"1=INSERIMENTO          2=VARIAZIONE             3=CANCELLAZIONE"
790 INPUTR:IFR<1ORR>3THEN790
795 FL=0:FF=0:ONRGOTO850,900,950
850 GOSUB1000:IFSW=0THEN870
```

```
860 GOSUB2000:CLOSE3:CLOSE2:PRINT"♥FINITO AGGIORNAMENTO"
863 PRINT#15,"S0:"+NF$
864 PRINT#15,"R0:"+NF$+"=TEMP":CLOSE15:STOP
870 IFI$(1)>LC$THEN875
872 IFI$(1)=LC$THENIFI$(2)>LN$THEN875
873 GOSUB1260:GOTO850
875 LC$=I$(1):LN$=I$(2):GOSUB1500
876 IFI$(1)=I1$(1)ANDI$(2)=I1$(2)THENPRINT"NOMI UGUALI":GOSUB1250:GOTO850
880 GOSUB1200:GOTO850
900 GOSUB1000:IFSW=0THEN930
920 GOTO860
930 IFI$(1)>LC$THEN935
932 IFI$(1)=LC$THENIFI$(2)>LN$THEN935
933 GOSUB1260:GOTO900
935 LC$=I$(1):LN$=I$(2):GOSUB2500
937 IFFL=1THEN900
940 GOSUB1200:GOTO900
950 PRINTD$(1);:INPUTI$(1):IFI$(1)="*"THEN860
953 PRINTD$(2);:INPUTI$(2):IFI$(1)>LC$THEN975
972 IFI$(1)=LC$THENIFI$(2)>LN$THEN975
973 GOSUB1260:GOTO950
975 LC$=I$(1):LN$=I$(2):GOSUB2500
980 IFFF=1ANDFL=0THEN860
985 GOTO950
1000 SW=0
1010 PRINT"] 1 ";D$(1);:INPUTI$(1)
1020 IFI$(1)="*"THENSW=1:RETURN
1030 FORK=2TO6:PRINTK;D$(K);:INPUTI$(K):NEXTK
1040 PRINT"♥CONFERMI S/N";:INPUTR$
1041 IFR$="S"THENRETURN
1043 PRINT"QUALE CAMPO ";:INPUT R
1044 IFR<1ORR>6THENPRINT"]";:GOTO1043
1045 PRINTD$(R);:INPUTI$(R):PRINT"]";
1046 FORK=1TO6:PRINTK;D$(K);I$(K):NEXTK:GOTO1040
1100 FORK=1TO6:INPUT#3,I$(K):FS=ST:GOSUB3000:NEXTK:RETURN
1200 FORK=1TO6:PRINT#2,I$(K);CH$;:GOSUB3000:NEXTK:RETURN
1250 GETA$:IFA$=""THEN1250
1251 RETURN
1260 PRINT"♥NOMI FUORI ORDINE":GOSUB1250:RETURN
1270 PRINT"♥MONTA DISCO DATI":RETURN
1280 INPUT"NOME FILE ";NF$:RETURN
1290 OPEN3,8,3,"0:"+NF$+",S,R":GOSUB3000:RETURN
1500 IFFF=1ANDFL=0THENRETURN
1501 IFFL<>0THEN1520
1505 GOSUB1555
1506 IFFS=64THENFF=1
1520 IFI1$(1)<I$(1)THEN1540
1530 IFI1$(1)=I$(1)THENIFI1$(2)<I$(2)THEN1540
1535 FL=1:RETURN
1540 GOSUB1600:FL=0
1545 IFFF=1THENRETURN
1550 GOTO1505
1555 FORK=1TO6:INPUT#3,I1$(K):FS=ST:GOSUB3000:NEXTK:RETURN
1600 FORK=1TO6:PRINT#2,I1$(K):GOSUB3000:NEXTK:RETURN
2000 IFFF=1ANDFL=0THENRETURN
2001 IFFL<>0THENFL=0:GOSUB1600:GOTO2003
2003 GOSUB1555:GOSUB1600:IFFS=64THENFF=1:RETURN
2004 GOTO2003
2500 IFFF=1ANDFL=0THEN2533
2501 IFFL<>0THEN2520
```

```
2505 GOSUB1555:IFFS=64THENFF=1
2520 IFI1$(1)<I$(1)THEN2540
2530 IFI1$(1)=I$(1)THENIFI1$(2)<I$(2)THEN2540
2532 IFI1$(1)=I$(1)ANDI1$(2)=I$(2)THENFL=0:RETURN
2533 FL=1:PRINT"NON TROVATO":RETURN
2540 GOSUB1600
2547 IFFF=1THEN2533
2550 GOTO2505
3000 INPUT#15,EN,EM$,ET,ES
3001 IFEN=0THEN RETURN
3002 PRINT"ERRORE DISCO"
3003 PRINTEN,EM$,ET,ES:STOP
```

Il programma è privo di commenti per renderlo il più corto possibile e poterlo usare anche senza espansioni di memoria; comunque esso gira con qualunque espansione. Nel programma si è prestata attenzione alla gestione di un file sequenziale trascurando altri aspetti come: contarighe per il prospetto, quadri video estetici. Si può creare un file sequenziale di indirizzi, fornendo i cognomi e i nomi in ordine alfabetico e senza ripetizioni; nel caso si hanno messaggi di avviso. Le funzioni sono:

1)creazione file ex novo;

2)aggiornamento file con aggiunte, modifiche e cancellazioni;

3)stampa file, con i dati su 4 righe: cognome-nome, telefono, indirizzo, CAP e città.

Per ogni nominativo si caricano i 6 campi elencati al punto 3). I campi sono di lunghezza variabile. In fase di aggiornamento si usa un file temporaneo di nome TEMP per ricopiare il file aggiornandolo e poi si usa la funzione RENAME per riassegnare al nuovo file il nome vecchio. Il nome da assegnare al file viene chiesto quando si crea. Nel programma si usa estensivamente la tecnica dei sottoprogrammi. Si fa ricorso ad alcuni flag per segnalare l'assenza del nominativo richiesto, la fine del file e per uscire dalla fase di input. Per uscire dalla fase di input si deve rispondere con * alla richiesta del cognome. Si fa notare l'uso della parola di stato ST per "fine-file" e della routine di errore.

```
LISTA ARCHIVIO ARCHI1

COGNOMEA    NOMEA
111111
VIA PRIMA 1
111111    PRIMACITTA'

COGNOMEF    NOMEF
222222
VIA SECONDA 222222
222222    SECONDACITTA'

COGNOMEZ    NOMEZ
33333333
VIA TERZA 3
33333    TERZACITTA'
```

Listato esempio prodotto dal programma

6.7 FILE RANDOM

Con il termine RANDOM, applicato a un file su disco, si intende un file nel quale si accede ai singoli record fisici (settori), fornendo al sistema l'indirizzo della traccia e del settore. Come si vede la parola "random" non è usata nel normale significato, che la lega a tecniche di randomizzazione. In realtà sarebbe più opportuno chiamare questa organizzazione "file a indirizzo fisico diretto". Il DOS fornisce una serie di istruzioni che consentono di gestire questi tipi di file. Prima di vedere il significato delle singole istruzioni, è bene soffermarsi a considerare le relazioni tra record logico e record fisico in un programma.
Il programmatore deve, come al solito studiare l'organizzazione del record logico necessario, configurato in campi, ma deve tener presente che la gestione dello stesso, nell'ambito del record fisico è a suo carico.
Si cercherà, in generale, di non complicare troppo le cose scegliendo per esempio, una lunghezza di record logico che sia un sottomultiplo del settore (mezzo settore, un quarto di settore) o che coincida con il settore (questo è il caso più semplice). Si possono, però, avere record logici che superano la dimensione di un settore; in questo caso il primo settore dovrà contenere l'indirizzo della traccia e del settore da leggere per poter proseguire.
Il primo problema da affrontare è quello della reperibilità di un certo record, cioè la sua identificazione. Nel record deve essere presente un campo che lo identifica. La situazione ideale è quella di avere un campo del record che contenga un numero, sul quale sia possibile operare con un algoritmo una trasformazione che dia luogo ad un numero che si trovi in progressione naturale rispetto agli altri. Il caso più semplice è quello della numerazione:1, 2, 3,...,n. Qualora i record siano stati registrati a partire da una traccia T e da un settore S uno dopo l'altro in sequenza, diciamo di chiave, potrà il programma stesso, applicando un algoritmo di calcolo, trovare in quale traccia e settore si trova l'ennesimo record del file. Di solito le cose non sono così semplici; per esempio, per formare un archivio clienti, si possono assegnare ai clienti dei numeri progressivi, man mano che si presentano, ma questa numerazione sicuramente non rifletterà l'ordine alfabetico per cognome e nome. In conseguenza, per sapere che il signor CLIENTE VERDI è il quindicesimo record logico del file, si dovrà consultare un elenco, possibilmente tenuto in ordine alfabetico, su carta. Questo numero 15 affidato al programma fa comparire dopo poco sul video il record logico desiderato. E' indubbiamente più comodo poter richiedere al programma: CLIENTE VERDI e ottenere il record senza passare per rubriche cartacee. Per ottenere questo si deve impostare in modo un pò più complicato la gestione del file, arrivare cioè all'organizzazione del file con indice. Si noti che con l'organizzazione a indice si occupa più spazio su disco e quindi si paga in termini di memoria di massa la comodità raggiunta.
Quale è il vantaggio rispetto al file sequenziale? Nell'organizzazione sequenziale per ottenere il quindicesimo record se ne devono leggere prima 14.

L'organizzazione dei file con indice consente la massima flessibilità; si ha un file

principale organizzato in modo RANDOM e un secondo file, indice del principale, organizzato in modo sequenziale. Il record logico del file principale può essere lungo quanto serve, ma il record logico del file indice deve essere il più corto possibile e deve contenere la chiave di identificazione, base per le ricerche, e l'indirizzo di traccia e settore del record fisico nel quale si trova (o inizia) il record corrispondente del file principale. Il file indice, sequenziale, può essere ordinato, in base alla chiave, secondo le necessità e usato per le ricerche. Qualora il file indice sia abbastanza corto, tanto da poter essere tenuto in memoria centrale del calcolatore durante le elaborazioni, si verifica la situazione ideale per il veloce reperimento delle informazioni. Infatti le ricerche in memoria sono molto veloci e basta una sola lettura da disco per trovare il record voluto. Se il file indice non può essere mantenuto tutto in memoria, viene letto a pezzi e si può eventualmente ricorrere a un secondo indice, contenibile in memoria, che permetta di caricare velocemente la parte del vero indice che interessa.

Dopo essersi impadroniti bene di queste tecniche, si può lavorare su organizzazioni anche complicate. Per esempio, si possono organizzare record logici con campi di lunghezza variabile, riportando prima di ogni campo la sua lunghezza.

I comandi del DOS che consentono la gestione dei file impegnano il canale 15 dei comandi e un altro canale con il relativo buffer. All'interno del buffer si usano le istruzioni di lettura e scrittura, già viste per i file sequenziali, e cioè: PRINT#, INPUT# e GET#. Queste istruzioni agiscono in modo sequenziale all'interno del buffer, ma è disponibile una istruzione che consente di "spostare il puntatore" e quindi di muoversi liberamente nel buffer.

I comandi DOS vengono impartiti con delle istruzioni PRINT#lfn,"stringa-comandi", dove "lfn" è il numero logico del file usato per aprire il canale 15 dei comandi. Negli esempi, come in precedenza, si userà 15 come "lfn". E' necessaria una OPEN particolare per aprire un canale ed avere a disposizione il buffer per lavorare; il formato è:

OPEN lfn,8,sa,"#"

dove lfn è il numero logico del file da usare nelle operazioni di lettura o scrittura;
8 è il numero della periferica disco;
sa è il numero del canale da impegnare.
Con la precedente istruzione non viene richiesto un particolare buffer tra quelli disponibili; è possibile richiedere uno tra gli 8 buffer della RAM del disco, facendo seguire al carattere "#" il numero del buffer richiesto. Qualora il buffer richiesto sia occupato si ottiene il messaggio di errore "NO CHANNELS". Lo stesso messaggio si ottiene anche se, senza richiedere un buffer particolare, si cerca di aprire più file di quelli consentiti. Dopo l'esecuzione del comando OPEN si può conoscere l'indirizzo del buffer allocato eseguendo una operazione GET# su "lfn"

usato nella OPEN, prima però di qualunque altra operazione di lettura o scrittura.

Si riporta l'elenco dei comandi DOS, da impartire, dopo l'apertura del canale 15, con una istruzione PRINT#15,"stringa-comandi" (si usa ancora 15 come "lfn").

COMANDI		Formato Stringa
Completi	Abbbreviati	
BLOCK-READ	B-R	"B-R:"ch,dr,t,s
BLOCK-WRITE	B-W	"B-W:"ch,dr,t,s
BLOCK-EXECUTE	B-E	"B-E:"ch,dr,t,s
BUFFER-POINTER	B-P	"B-P:"ch,p
BLOCK-ALLOCATE	B-A	"B-A:"dr,t,s
BLOCK-FREE	B-F	"B-F:"dr,t,s
Memory-Write	M-W	"M-W"adl/adh/nc/data
Memory-Read	M-R	"M-R"adl/adh
Memory-Execute	M-E	"M-E"adl/adh
USER	U	"Ui:parms"

dove:

ch	numero canale, sa della OPEN
dr	numero unità=0
t	numero traccia: da 1 a 35
s	numero settore: da 0 a 20, o meno, in dipendenza dal gruppo cui appartiene la traccia
p	posizione del puntatore nel buffer
adl	byte basso del blocco di memoria
adh	byte alto del blocco di memoria
nc	numero caratteri: da 1 a 34
data	dato attuale in notazione esadecimale: si deve usare la funzione CHR$(codice decimale equivalente)

i	numero di riferimento della tabella USER (riportata più avanti)

parms	parametri associati a U

Nella stringa dei comandi è obbligatorio usare le abbreviazioni per i 3 comandi Memory, per gli altri è opzionale.
Qualora i parametri siano variabili, esse non possono essere scritte tra gli apici, mentre se sono costanti si. All'interno degli apici si possono usare come separatori: freccia a destra, spazio, virgola; fuori dagli apici, invece, solo il punto e virgola.

E' consigliabile inizializzare il dischetto prima delle operazioni di lettura e scrittura per garantirsi un buon allineamento e una corretta registrazione della BAM in memoria.

Il comando BLOCK-READ trasferisce il settore specificato nel buffer assegnato al momento della OPEN. Dopo il trasferimento il puntatore si trova nella posizione 0 del buffer. Con il puntatore in questa posizione una operazione di lettura genera il segnale di End-Of-Identify (EOI) che pone il valore 64 nella Status Word del calcolatore (ST=64). Prima di proseguire la lettura si deve porre il puntatore nella posizione 1 del buffer.
La stringa-comando: "B-R:3,0,5,7" legge nel buffer associato al canale 3 il settore 7 della traccia 5.

Il comando BLOCK-WRITE trasferisce il contenuto del buffer nel blocco indicato da traccia e settore. Prima del trasferimento il puntatore viene posto nella posizione 0 del buffer, dopo il trasferimento viene lasciato nella posizione 1 del buffer.
La stringa comando: "B-W:3,0,5,7" scrive il contenuto del buffer associato al canale 3 nel settore 7 della traccia 5.

Il comando BLOCK EXECUTE legge un blocco dal disco nel buffer associato al canale, e manda in esecuzione il programma in linguaggio macchina contenuto nel blocco a partire dalla posizione 0 del buffer. Per un corretto funzionamento il programma contenuto deve terminare con una istruzione di "return da sottoprogramma (RTS)". Il programma viene eseguito dal processore del disco.
La stringa comando: "B-E:4,0,3,5" legge nel buffer associato al canale 4 il settore 5 della traccia 3 e va ad eseguire il programma partendo dal byte di posizione 0.

Il comando BUFFER-POINTER sposta il puntatore all'interno del buffer consentendo di accedere a particolari record logici o campi all'interno di esso.
La stringa comando: "B-P:3,1" sposta nel buffer associato al canale 3 il puntatore nella posizione 1 del buffer (le posizioni vanno da 0 a 255).

Il comando BLOCK-ALLOCATE consente di modificare la BAM rendendo indi-

sponibile un particolare settore. Si ricordi che la BAM viene riscritta sul dischetto quando si chiude un file in scrittura o un canale associato a un file ad accesso diretto. Nel comando è indicato un indirizzo di traccia e settore; dopo l'esecuzione, se il settore richiesto era già occupato, il canale degli errori segnala un errore "NO BLOCK" (codice errore 65) e anche un indirizzo di traccia e settore disponibili. Qualora il disco sia pieno si trova zero nell'indirizzo di traccia e settore. Per allocare il settore indicato si deve ripetere l'operazione usando il nuovo indirizzo.
La stringa comando: "B-A:0,8,5" alloca il settore 5 della traccia 8 se il blocco è libero.

Il comando BLOCK-FREE serve per liberare un blocco occupato e quindi aggiorna la BAM.
La stringa comando: "B-F:0,5,8" libera il settore 8 della traccia 5.
Questo comando e il precedente, che lavorano sulla BAM, possono essere usati anche se il canale per l'accesso diretto al file non è aperto. Si deve però fare attenzione affinchè avvenga la riscrittura della BAM su disco, e quindi usare questi comandi mentre è aperto un canale, in modo che alla sua chiusura avvenga l'aggiornamento del disco.

I 3 comandi che iniziano con la parola Memory consentono di accedere al massimo a 34 byte della memoria disco; essi si possono scrivere solo in forma abbreviata e nella stringa non possono comparire i due punti. Per i parametri numerici si deve usare la funzione CHR$ che trasmette i numeri esadecimali byte a byte; i byte delle istruzioni vengono scritti in decimale. L'indirizzo del byte iniziale deve essere trasmesso diviso in 2 byte passando prima il byte basso e poi quello alto; per esempio, l'indirizzo decimale 1794, che in esadecimale è 0702, deve essere visto come 02+07*256 e passato con CHR$(02) e CHR$(07).

Il comando Memory-Write consente di scrivere nella RAM del DOS particolari routine in linguaggio macchina, che possono poi essere eseguite con il comando Memory-Execute. La stringa comando

"M-W"CHR$(2)CHR$(7)CHR$(5)CHR$(169)CHR$(5)CHR$(76)CHR$(18)CHR$(7)

scrive all'indirizzo decimale 1794 (0702H) 5 byte e precisamente:

```
169 A9H
  5 05H
 76 4CH
 18 12H
  7 07H
```

cioè le istruzioni

```
LDA #05H
JMP 0712H
```

Il comando Memory-Read rende accessibile un byte della memoria DOS, quello il cui indirizzo compare nel comando. Si può così accedere a variabili del DOS o a caratteri del buffer. A questo comando deve seguire una istruzione GET#15, che va a leggere attraverso il canale comandi il byte. Si noti che la lettura avviene tramite il canale 15, canale comandi/errori, che quindi viene usato non nel normale modo (INPUT#15) usato per leggere le 4 variabili del messaggio di errore. Dopo questo comando il canale 15 non può essere interrogato con INPUT nel solito modo fino a dopo l'esecuzione di un comando DOS non di tipo Memory. La stringa comando:

"M-R"CHR$(128)CHR$(00)

consente di accedere al byte di indirizzo decimale 128 (0080H).

Il comando Memory-Execute consente di eseguire una routine contenuta nella memoria del DOS che inizia all'indirizzo del byte passato dall'istruzione. Per un corretto svolgimento del lavoro la routine deve terminare con la istruzione RTS (codice 60H). La stringa comando:

"M-E"CHR$(02)CHR$(07)

manda in esecuzione una routine in linguaggio macchina che inizia all'indirizzo decimale 1794 (0702H).

I comandi di tipo USER consentono di saltare, tramite la tabella dei salti che segue, a particolari routine del DOS. Il carattere che segue la lettera U fa puntare a un particolare indirizzo.

TABELLA DEI SALTI		
Prima Definizione	Definizione Alternativa	Significato comando
U1	UA	sostituisce BLOCK-READ
U2	UB	sostituisce BLOCK-WRITE
U3	UC	salto a 1280 (0500H)
U4	UD	salto a 1283 (0503H)
U5	UE	salto a 1286 (0506H)
U6	UF	salto a 1289 (0509H)
U7	UG	salto a 1292 (050CH)
U8	UH	salto a 1295 (050FH)
U9	UI	salto a 65530 (FFFAH)
U0	UJ	salto alla routine di accensione

I primi due comandi, che sostituiscono quelli di BLOCK-READ e di BLOCK-WRITE sono molto utili, dato che il primo (U1), forzando il puntatore del buffer a 255, consente di accedere a tutto il settore, indipendentemente dalla posizione del puntatore al momento della scrittura, mentre il secondo (U2) scrive il blocco senza cambiare la posizione del puntatore. I parametri dei due comandi sono identici a quelli di B-R e di B-W.
I comandi da U3 a U9 consentono di mandare in esecuzione routine memorizzate con il comando M-W ai particolari indirizzi elencati. Il comando U0 fa saltare alla routine di Power-up.

6.7.1 ESEMPIO ARCHIVIO RANDOM

Segue come esempio un programma che gestisce un file RANDOM con indice primario e, volendo, anche secondario.

```
1 REM FILERANDOM
20 SP$="                              "
30 CH$=CHR$(13):LIM$=CHR$(99)+CHR$(99)+CHR$(99)
40 DIMD$(14):DIMY$(14):DIML(14)
50 D$(1)="C:"
60 D$(2)="N:"
70 D$(3)="I:"
80 D$(4)="L:"
90 D$(5)="P:"
100 D$(6)="T:"
110 D$(7)="LN:"
120 D$(8)="DN:"
130 D$(9)="TS:"
140 D$(10)="OA:"
150 D$(11)="OP:"
160 D$(12)="SC:"
170 D$(13)="N1:"
180 D$(14)="N2:"
190 L(1)=20:L(2)=15:L(3)=30:L(4)=25:L(5)=4
200 L(6)=10:L(7)=20:L(8)=8:L(9)=20:L(10)=20:L(11)=20:L(12)=8
210 L(13)=20:L(14)=20
220 OPEN15,8,15
230 PRINT"❏GESTIONE ARCHIVIO":PRINT
250 PRINT"1=INIZIO EX-NOVO"
260 PRINT"2=AGGIORNAMENTO"
270 PRINT"3=LISTA PRINCIPALE"
280 PRINT"4=CREAZ.IND.SEC."
290 PRINT"5=LISTA SECONDARIA"
300 PRINT"9=FINE"
310 INPUT"COSA SCEGLI ";X
320 IFX<1ORX>5ANDX<>9THEN310
325 PRINT:GOSUB1180
330 PRINT"MONTA DISCO DATI"
340 GETR$:IFR$=""THEN340
343 IFX=1THEN380
345 PRINT#15,"I"
```

```
350 GOSUB960:N=K:PRINT"PRESENTI ";K;" RECORD"
360 PRINT"DATA ULTIMO AGG.      ";G1$;"/";M1$;"/";A1$
370 IFX<>2THEN400
380 PRINT"QUANTI RECORD ";:INPUT N
400 DIMC$(N),T%(N),S%(N)
410 IFX<>1THENGOSUB1050
420 IFX=9THEN2840
430 ONXGOTO1270,1560,2360,2600,2730
440 STOP
450 REM INGRESSO DATI
460 PRINT"]INGRESSO DATI"
470 PRINT"PER USCIRE $ PER COGN.SE MANC. DATI METTI 0"
490 PRINTD$(1);:INPUTY$(1)
500 IFY$(1)="$"THENW=1:RETURN
510 FORJ=2TO14:PRINTD$(J);:INPUTY$(J):NEXTJ
540 FORJ=1TO14:Y$(J)=LEFT$(Y$(J)+SP$,L(J)):NEXTJ:RETURN
580 REM INDICE PRINCIPALE
590 PRINT#15,"S0:INDI1":PRINT#15,"I":OPEN10,8,10,"INDI1,S,W":GOSUB1120
600 FORJ=1TOK:PRINT#10,C$(J);CH$;T%(J);CH$;S%(J);CH$;:GOSUB1120:NEXTJ
630 CLOSE10:RETURN
640 REM RECORD E ELEM. INDICE
650 FORJ=1TO14:PRINT#11,Y$(J);CH$;:GOSUB1120:NEXTJ:RETURN
700 REM TRACCIA E SETTORE
710 PRINT#15,"B-A:"0;T;S
720 INPUT#15,EN,EM$,ET,ES
730 IFEN=0THENRETURN
740 IFEN<>65THEN1140
750 T=ET:S=ES:GOTO710
760 REM PUNTATORE
770 PRINT#15,"B-P:"11;1:GOSUB1120:RETURN
800 REM SCRITTURA
810 PRINT#15,"U2:"11;0;T;S:GOSUB1120:RETURN
840 REM LETTURA
850 PRINT#15,"I":OPEN11,8,11,"#":GOSUB1120
870 PRINT#15,"U1:"11;0;T;S:GOSUB1120:GOSUB770
910 FORJ=1TO14:INPUT#11,Y$(J):GOSUB1120:NEXTJ:CLOSE11:RETURN
960 REM LETT.DATI DISCO
970 OPEN11,8,11,"#":GOSUB1120:PRINT#15,"U1:"11;0;1;0:GOSUB1120
990 GOSUB760:INPUT#11,R$,K,T,S:GOSUB1120
1010 G1$=LEFT$(R$,2):M1$=MID$(R$,3,2):A1$=RIGHT$(R$,2):CLOSE11:TT=T:SS=S:RETURN
1050 REM LETT. INDICE
1060 OPEN10,8,10,"INDI1,S,R":GOSUB1120
1070 FORJ=1TOK:INPUT#10,C$(J),T%(J),S%(J):GOSUB1120:NEXTJ:CLOSE10:RETURN
1120 INPUT#15,EN,EM$,ET,ES
1130 IFEN=0THENRETURN
1140 PRINT"XERRORE DISCO":PRINTEN,EM$,ET,ES:CLOSE15:STOP
1180 REM DATA DISCO
1190 PRINT"DATA PER DISCO"
1200 INPUT"  GG,MM,AA███████████";G$,M$,A$:RETURN
1210 REM SCRITT. IND. SEC.
1220 PRINT#15,"S:INDI2":PRINT#15,"I":OPEN10,8,10,"INDI2,S,W"
1230 FORJ=1TOK:PRINT#10,C$(J);CH$;T%(J);CH$;S%(J);CH$;:GOSUB1120:NEXTJ:CLOSE10:R
ETURN
1250 GETR$:IFR$=""THEN1250
1251 RETURN
1270 REM INIZIALIZZAZIONE DISCO
1310 PRINT"NOME DISCO";:INPUTN$:T=1:S=0
1330 PRINT#15,"N0:"+N$+",99"
1335 CLOSE15:OPEN15,8,15:PRINT#15,"I"
```

```
1350 REM DATA AGGIORNAMENTO E NUM. RECORD
1370 OPEN11,8,11,"#":GOSUB1120:GOSUB710:GOSUB770
1400 K=0: PRINT#11,G$+M$+A$;CH$;K;CH$;2;CH$;0;CH$;:GOSUB1120
1440 PRINT#15,"U2:"11;0;T;S:GOSUB1120:K=1:W=0:T=2:S=0
1450 GOSUB450
1460 IFW=1THENK=K-1:CLOSE11:GOTO2840
1470 GOSUB700:GOSUB760:GOSUB640:C$(K)=Y$(1)+Y$(2)
1510 T%(K)=T:S%(K)=S:GOSUB800:K=K+1
1540 IFK>NTHENK=K-1:PRINT"FINITO SPAZIO ASSEGNATO":CLOSE11:GOTO2840
1550 GOTO1450
1560 REM AGGIORNAMENTO
1570 PRINT"⊐AGGIORNAMENTO ARCHIVIO"
1580 PRINT"1=CORREZIONE"
1590 PRINT"2=AGGIUNTA ELEM."
1600 PRINT"3=CANCELL.ELEM."
1610 PRINT"9=FINE"
1620 INPUT"COSA SCEGLI ";X:IFX=9THEN2840
1640 IFX<1ORX>3THEN1570
1650 IFX=2THEN2070
1660 IFX=3THEN2140
1670 REM CORREZIONE
1680 PRINT"⊐";D$(1);:INPUTY$(1)
1690 IFY$(1)="$"THEN1570
1700 PRINTD$(2);:INPUTY$(2)
1710 Y$(1)=LEFT$(Y$(1)+SP$,L(1))
1720 Y$(2)=LEFT$(Y$(2)+SP$,L(2))
1730 INPUT"QUALE OCCORRENZA ";X
1740 IFX<=0THEN1730
1750 FORJ=1TOK:IFC$(J)=Y$(1)+Y$(2)THEN1790
1770 NEXTJ
1780 J=J-1:PRINT"NON TROVATO ";Y$(1);"  ";Y$(2)
1785 GOSUB1250:GOTO1680
1790 IFX<>1THENX=X-1:GOTO1770
1800 T=T%(J):S=S%(J)
1810 I=J:J=K:NEXTJ
1820 GOSUB840
1821 J1=1:J2=7:GOSUB1830
1823 J1=8:J2=14:GOSUB1830
1825 GOTO1980
1830 PRINT"⊐";:FORJ=J1TOJ2:PRINTJ;" ";D$(J);" ";Y$(J):NEXTJ
1870 PRINT"QUALE CAMPO (0 USCITA)"
1880 INPUTX
1885 IFX=0THEN1930
1890 IFX>J2ORX<J1THENPRINT"⊤⊤":GOTO1880
1910 INPUTY$(X):Y$(X)=LEFT$(Y$(X)+SP$,L(X))
1920 PRINT"⊤⊤⊤":GOTO1880
1930 PRINT"⊐";:FORJ=J1TOJ2:PRINTJ;" ";D$(J);" ";Y$(J):NEXTJ
1960 INPUT"ЖTUTTO BENE ";X$:IFX$<>"S"THEN1870
1961 RETURN
1980 PRINT#15,"I":OPEN11,8,11,"#":GOSUB760:GOSUB640
2020 C$(I)=Y$(1)+Y$(2):T%(I)=T:S%(I)=S
2040 GOSUB800:CLOSE11:GOTO1670
2070 REM AGGIUNTA
2080 OPEN11,8,11,"#":GOSUB1120:PRINT"⊐AGGIUNTA":PRINT"PRESENTI ";K;" RECORD"
2110 PRINT"PUOI AGGIUNGERE ";N-K;" RECORD":GOSUB1250
2120 T=TT:S=SS:W=0:K=K+1:GOTO1450
2140 REM CANCELLAZIONE
2150 PRINT"ЖCANC. ELEM.":M=0
2170 PRINTD$(1);:INPUTY$(1)
2180 IFY$(1)="$"THEN2320
```

```
2190 PRINTD$(2);:INPUTY$(2)
2200 Y$(1)=LEFT$(Y$(1)+SP$,L(1))
2210 Y$(2)=LEFT$(Y$(2)+SP$,L(2))
2220 FORJ=1TOK:IFC$(J)=Y$(1)+Y$(2)THEN2260
2240 NEXTJ
2250 PRINT"NON TROVATO ";Y$(1);" ";Y$(2)
2255 GOSUB1250:GOTO2170
2260 X=L(1)+L(2):C$(J)=LEFT$(LIM$+SP$+SP$,X)
2270 M=M+1:T=T%(J):S=S%(J)
2290 PRINT#15,"B-F:"0;T;S:GOSUB1120:GOTO2170
2320 REM SIST. INDICE
2330 GOSUB2980:K=K-M:GOTO2890
2360 REM LISTA
2370 T$="LISTA PER INDICE PRIM."
2380 PRINT"ACCENDI STAMPANTE      PREMI UN TASTO"
2390 GETR$:IFR$=""THEN2390
2400 PRINT"♥";T$:OPEN4,4
2430 PRINT#4:PRINT#4:PRINT#4,T$
2440 FORJ=1TO10:PRINT#4:NEXTJ
2450 L=2:FORI=1TOK:T=T%(I):S=S%(I):GOSUB840
2490 PRINT#4,Y$(1);"  ";Y$(2)
2500 PRINT#4,Y$(3);"  ";Y$(4);"  ";Y$(5)
2510 FORM=6TO14:PRINT#4,D$(M);Y$(M):NEXTM
2540 PRINT#4:PRINT#4
2550 IFL=5THENPRINT#4:L=0
2560 L=L+1:NEXTI
2580 CLOSE4:GOTO2950
2600 REM IND. SEC.
2610 PRINT"♥INDICE SECONDARIO"
2620 INPUT"QQUALE CAMPO ";X$
2630 X=VAL(X$):IFX<2ORX>14THENGOTO2620
2640 GOSUB580
2650 FORI=1TOK:T=T%(I):S=S%(I):GOSUB840:C$(I)=Y$(X):NEXTI
2700 GOSUB1210:GOSUB1050
2710 PRINT"FINITO IND. SEC."
2720 GOTO2950
2730 REM LISTA IND. SEC.
2740 PRINT"♥LISTA IND. SEC."
2750 OPEN10,8,10,"INDI2,S,R":GOSUB1120
2760 FORJ=1TOK:INPUT#10,C$(J),T%(J),S%(J):GOSUB1120:NEXTJ
2790 CLOSE10:GOSUB3120:GOSUB1210
2820 T$="LISTA IND. SEC. "
2830 GOTO2380
2840 REM CHIUSURA
2850 PRINT"CHIUSURA ARCHIVIO"
2860 PRINT"IND. PRINC. DA ORDIN. ":INPUTR$
2880 IFR$="S"THENGOSUB2980
2890 GOSUB580:OPEN11,8,11,"#":GOSUB1120
2910 PRINT#15,"B-P:"11,1
2920 PRINT#11,G$+M$+A$;CH$;K;CH$;T;CH$;S;CH$;:GOSUB1120
2930 PRINT#15,"U2:"11;0;1;0:GOSUB1120:CLOSE11
2950 PRINT"FINITO AGGIORNAMENTO"
2960 PRINT"SONO PRESENTI ";K;" RECORD"
2970 CLOSE15:STOP
2980 REM ORDINAMENTO CRESCENTE
3000 L=K-1
3010 W=0
3020 FORJ=1TOL
3030 IFC$(J)<=C$(J+1)THEN3080
3040 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
```

```
3050 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
3060 X=S%(J):S%(J)=S%(J+1):S%(J+1)=X
3070 W=1
3080 NEXTJ
3090 IFW=0THENRETURN
3100 IFL=1THENRETURN
3110 L=L-1:GOTO3010
3120 REM ORDINAMENTO DECRESCENTE
3140 L=K-1
3150 W=0
3160 FORJ=1TOL
3170 IFC$(J)>=C$(J+1)THEN3220
3180 R$=C$(J):C$(J)=C$(J+1):C$(J+1)=R$
3190 X=T%(J):T%(J)=T%(J+1):T%(J+1)=X
3200 X=S%(J):S%(J)=S%(J+1):S%(J+1)=X
3210 W=1
3220 NEXTJ
3230 IFW=0THENRETURN
3240 IFL=1THENRETURN
3250 L=L-1:GOTO3150
```

Il programma FILERANDOM è organizzato per gestire un file random al quale si può assegnare il nome desiderato al momento della creazione, come nome per il disco, che il sistema va a scrivere nella traccia 18 settore 0 subito dopo la BAM, in fase di formattazione (linee 1270/1335). Tale file non compare nella Directory, mentre in essa compaiono i 2 file sequenziali che costituiscono l'indice primario e secondario (se desiderato) del file random. L'indice primario si chiama INDI1 e pensa il programma a scriverlo e riscriverlo quando necessario, cancellando la copia precedente. L'indice secondario INDI2 viene creato se richiesto usando un campo prescelto del record, e può essere rifatto tutte le volte che si vuole cambiando il campo (dal secondo al quattordicesimo). INDI1 viene ordinato in modo crescente, mentre INDI2 viene ordinato in modo decrescente. I campi del record sono trattati tutti come stringhe e sono tutti di uguale lunghezza, con l'eventuale aggiunta di spazi di riempimento; per questa ragione si deve fare attenzione affinchè eventuali campi numerici siano tutti lunghi uguali, ponendo in caso degli zeri a sinistra, altrimenti l'ordinamento non risulta buono. Il primo indice è formato usando i primi due campi concatenati (Cognome + Nome). Il record logico coincide, in questo caso, con il record fisico e risulta di 254 caratteri compresi i separatori tra i campi, carattere RETURN (CHR$(13)). Ai campi sono stati assegnati nomi abbreviati per esigenze di spazio sul video; si vedano le linee da 40 a 210 dove, dopo aver dimensionato a 14 (numero dei campi) i 3 vettori D$ per le descrizioni, Y$ per i dati e L per le lunghezze dei campi, si assegnano i nomi agli elementi di D$. Con questa impostazione non sarà difficile modificare il programma secondo le proprie esigenze cambiando i nomi ai campi e anche il loro numero e le loro lunghezze; attenzione però, se si cambia il 14, si devono ricercare nel programma tutti i 14, nei cicli, e modificarli. I campi sono (cr sta per Return):

1	cognome	C:	20 car. + cr
2	nome	N:	15 car. + cr
3	indirizzo	I:	30 car. + cr
4	località	L:	25 car. + cr
5	provincia	P:	4 car. + cr
6	telefono	T:	10 car. + cr
7	luogo nascita	LN:	20 car. + cr
8	data nascita	DN:	8 car. + cr
9	titolo studio	TS:	20 car. + cr
10	occ. attuale	OA:	20 car. + cr
11	occ. preced.	OP:	20 car. + cr
12	stato civile	SC:	8 car. + cr
13	nota 1	N1:	20 car. + cr
14	nota 2	N2:	20 car. + cr

Il programma occupa circa 6000 byte e quindi non gira senza espansioni di memoria. Inoltre occorre parecchia memoria per INDI1, che viene utilizzato sotto forma di tabella costituita da 3 vettori (dimensionamento alla linea 400 in base al numero N dei record richiesti): C$ per la chiave, T% per la traccia e S% per il settore. Per gestiire 200 record occorrono per INDI1 in memoria circa 8000 byte. Sul disco ogni record del file random occupa 1 settore più circa 40 byte per il record del file sequenziale INDI1 e circa 30 byte per il record del file sequenziale INDI2.

Il menù principale del programma è il seguente:

```
GESTIONE ARCHIVIO
1=INIZIO EX-NOVO
2=AGGIORNAMENTO
3=LISTA PRINCIPALE
4=CREAZ.IND.SEC.
5=LISTA SECONDARIA
9=FINE
```

mentre per la funzione AGGIORNAMENTO si ha il seguente menù:

```
AGGIORNAMENTO ARCHIVIO
1=CORREZIONE
2=AGGIUNTA ELEM.
3=CANCELL.ELEM.
9=FINE
```

L'allocazione dei settori per il file random si ottiene sfruttando la funzione B-A del DOS (linee da 700 a 750): si passa un valore per T e S e, se il settore è occupato, la

funzione fornisce il nuovo indirizzo che viene riallocato. Quando si cancella un record, lo spazio viene liberato usando la funzione B-F. In realtà l'indirizzo di traccia e settore fornito dalla B-A dovrebbe essere controllato per evitare di invadere la traccia 18. Nel programma questo non è stato fatto, dato che, partendo dal settore 0 della traccia 2 e usando al massimo 200 settori, non si arriva alla traccia 18.
Nel settore 0 della traccia 1 vengono sempre riscritti: la data di aggiornamento , il numero dei record presenti e l'indirizzo dell'ultimo settore usato. Il file random inizia alla traccia 2, settore 0 (linea 1440). Nel programma, dopo le assegnazioni iniziali e la scelta dell'operazione, sono localizzati dalla linea 450 alla 1251 un buon numero di sottoprogrammi utilizzati nel seguito. La funzione INIZIALIZZAZIONE parte dalla linea 1270, quella di AGGIORNAMENTO dalla linea 1560, quella di LISTA dalla linea 2360, quella di CREAZ.IND.SEC dalla linea 2600 e quella di LISTA SECONDARIA dalla linea 2730. La parte FINE si trova a partire dalla linea 2840; è importante che venga ordinato e riscritto l'indice principale quando ci sono state variazioni nel file. Nella parte CORREZIONE della fase AGGIORNAMENTO si è dovuto spezzare in due parti l'evidenziazione sul video e la modifica, dato che non potevano entrare 14 dati tutti insieme.
La routine di ordinamento crescente per INDI1 si trova da 2980 a 3110; quella di ordinamento decrescente per INDI2 da 3120 a 3250. Si tratta del metodo di ordinamento a bolle, che non risulta molto veloce in presenza di molti record. Tali routine potrebbero essere sostituite con altre più efficienti. Nel caso di cancellazione di record vengono posti a un valore alto i primi 3 caratteri della chiave dell'indice principale, così nell'ordinamento i relativi record vanno verso il fondo e non vengono riscritti. I listati non hanno una forma particolarmente curata: vengono stampati sulla prima riga i primi 2 campi, sulla seconda i 3 seguenti e sulle altre gli altri campi, uno per riga.

Per ovviare al fatto che i file RANDOM non vengono registrati nella Directory si può procedere così:
.1)stabilire quanti settori servono per il file random;
.2)aprire un file di tipo USER (che è sequenziale) DUMMY (FALSO) al solo scopo di occupare il numero desiderato di settori, scrivendo in ogni settore dei caratteri di riempimento, e ottenendo così di avere il concatenamento dei settori (primi due byte con indirizzo di traccia e settore, e traccia a zero per l'ultimo) fatto dal sistema, e poi chiuderlo;
.3)andare a leggere nella Directory quale è il primo indirizzo assegnato al file (si veda il programma CONT/DISCO nel paragrafo dei programmi di utilità) e conservarlo;
.4)rileggere in modo random il primo settore del file di tipo USER e prelevare dai primi 2 byte l'indirizzo del prossimo settore da occupare e preparare una tabella in memoria con gli indirizzi di tutti i settori utilizzati dal file USER. Leggere tutti i settori del file con lo stesso sistema e completare la tabella in memoria.

.5)scrivere il file RANDOM riutilizzando i settori già allocati per il file sequenziale, senza distruggere il contenuto dei primi 2 byte e seguendo la catena dei concatenamenti, ma non usare la funzione B-A.
Per lavorare meglio, nella fase di rilettura del file USER, può essere già preparato in memoria un eventuale indice per il file, registrando gli indirizzi di traccia e settore che vengono via via utilizzati dal file sequenziale. Tale indice potrà poi essere completato con i valori delle chiavi nella fase di creazione del file RANDOM. Se si lavora così i dischi dati sono più sicuri; su di essi può essere eseguita la funzione VALIDATE senza danneggiare il file RANDOM che ha una "falsa registrazione" nella Directory con il file di tipo USER.

6.8 FILE RELATIVI

Con le parole "File Relativi" si intende una organizzazione nella quale si accede al Record Logico fornendo il numero d'ordine del record nel file: 1 per il primo record, 5 per il quinto record, e così via. Il VIC 20 consente di gestire file relativi con record logici di lunghezza fissa minore o uguale a 254 caratteri, compresi i caratteri separatori (CHR$(13)). La lunghezza del record può essere scelta a piacere e non deve necessariamente essere un sottomultiplo della lunghezza di un settore; sono gestiti infatti, come per i file sequenziali, anche i record logici spezzati (spanned) in due settori. Il sistema mantiene su disco delle tabelle, chiamate SIDE SECTOR, nelle quali sono registrati i puntatori ai diversi settori del file. Sono consentite al massimo 6 tabelle e ognuna può contenere i riferimenti a 120 blocchi fisici; si arriva così a 720 blocchi (settori) al massimo per un file (nel VIC tale numero è sovrabbondante, data la capacità del floppy).
Il file relativo deve essere preesteso al momento della creazione scrivendo record "vuoti", cioè contenenti caratteri di puro riempimento. In seguito si ha la massima libertà di scrivere o riscrivere i record con i caratteri "veri". La routine di errore, in fase di estensione del file, dà ERRORE 50, ma si deve procedere ugualmente.
I file relativi vengono registrati nella directory del disco, e possono essere cancellati con il comando SCRATCH.
Oltre allo spazio disco necessario per i record del file, viene occupato lo spazio necessario per i SIDE SECTOR.

Per gestire i file relativi viene sfruttato il canale 15 dei comandi e degli errori, più un canale per il file. I comandi da usare sono i seguenti:

OPEN 15,8,15,"I"

per aprire il canale comandi e inizializzare il dischetto

OPEN lfn, 8, CH, fn + ",L," + CHR$(LU)

dove: lfn è il numero logico del file da usare nei comandi di lettura e scrittura

CH è il numero del canale scelto per comunicare

fn è il nome del file

L è un carattere richiesto

LU è la lunghezza del record logico e va fornito come numero decimale con la funzione CHR$

PRINT#15,stringa-comando

dove:"stringa-comando" fornisce i parametri necessari a localizzare il record logico ed è costruita così:

"P" + CHR$(lfn) + CHR$(LO) + CHR$(HI) + CHR$(BI)

con: lfn numero logico del file

LO e HI sono rispettivamente il byte basso (LO) e alto (HI) ottenuti operando sul numero del record con il seguente algoritmo:
HI = INT(numrec/256)
LO = numrec - HI*256

BI è il puntatore al byte nel record: 1 per accedere al primo campo del record, altro per accedere a altri campi (tale puntatore avanza automaticamente se in un comando di lettura o scrittura si lavora su più campi)

PRINT#lfn, dati

dove: lfn è il numero logico del file

"dati" sono le variabili e i loro separatori; vale la regola che in fase di scrittura se non si termina con ";" il sistema aggiunge un RETURN che può andare a disturbare la lunghezza prefissata per il record logico

CLOSE lfn per chiudere il file logico

CLOSE 15 per chiudere il canale comandi

La routine di errore deve ammettere il codice 0 e il codice 50 come validi.

Seguono 4 programmi esempio: REL1, REL2, REL3 e REL4.

```
1 REM REL1
2 REM CREAZIONE FILE RELATIVO
3 REM PREESTENSIONE FILE
4 REM 30 RECORD DI 21 CARATTERI
5 REM COMPRESO RETURN FINALE
100 DR$="0":NF$=DR$+":PROVA REL,L,"
105 CH=2:LF=2:RC$="..........":RC$=RC$+RC$
110 BI=1:NR=30:LU=21
120 OPEN15,8,15,"I"
130 OPENLF,8,CH,NF$+CHR$(LU)
140 GOSUB1000
150 FORX=NRTO1STEP-1
160 HI=INT(X/256):LO=X-HI*256
170 C$="P"+CHR$(LF)
180 C$=C$+CHR$(LO)+CHR$(HI)
190 C$=C$+CHR$(BI)
200 PRINT#15,C$:GOSUB1000
210 PRINT#LF,RC$:GOSUB1000
220 NEXTX:CLOSELF:GOSUB1000:CLOSE15:END
1000 INPUT#15,EN,EM$,ET,ES
1010 IFEN=0OREN=50THENRETURN
1020 PRINTEN;EM$;ET;ES
1030 STOP:RETURN
```

Questo programma serve per creare e preestendere un file relativo di nome PROVA REL con 30 record logici di 21 caratteri ciascuno; usa il numero logico 2 per il file e il canale 2 per lavorare. La routine di errore accetta codice 0 e 50 come validi. Inizialmente i record vengono riempiti con 20 puntini più il RETURN.

```
1 REM REL2
2 REM SCRITTURA RECORD NEL FILE RELATIVO
3 REM OGNI RECORD 2 CAMPI
4 REM NOME DI 10 CARATTERI
5 REM TELEFONO DI 9 CARATTERI
6 REM LUNGHEZZA RECORD=10+1+9+1=21
100 DR$="0":NF$=DR$+":PROVA REL,L,"
105 LU=21:CH=2:LF=2:B$="          "
110 BI=1:NR=30
113 OPEN15,8,15,"I"
115 OPENLF,8,CH,NF$+CHR$(LU)
117 GOSUB1000
120 PRINT"NOME=* PER USCIRE":PRINT
123 INPUT"NOME(10)";N$
125 IFN$="*"THEN250
127 N$=LEFT$(N$+B$,10)
130 INPUT"TEL. (9)";T$:T$=LEFT$(T$+B$,9)
140 INPUT"RECORD ";R
150 IFR=0ORR>NRTHENPRINTCHR$(145);:GOTO140
190 HI=INT(R/256):LO=R-HI*256
200 C$="P"+CHR$(LF)
210 C$=C$+CHR$(LO)+CHR$(HI)
220 C$=C$+CHR$(BI)
```

```
230 PRINT#15,C$:GOSUB1000
240 PRINT#LF,N$;CHR$(13);T$:GOSUB1000
245 GOTO120
250 CLOSELF:GOSUB1000:CLOSE15:END
1000 INPUT#15,EN,EM$,ET,ES
1010 IFEN=0OREN=50THENRETURN
1020 PRINTEN;EM$;ET;ES
1030 STOP:RETURN
```

Questo programma serve per scrivere record veri nel file; si possono scrivere i record desiderati e uscire rispondendo con * al primo campo. I campi del record logico sono 2: NOME di 10 caratteri e TELEFONO di 9; la lunghezza del record è 10+1+9+1=21. I campi vengono resi di lunghezza fissa con aggiunta di spazi.

```
1 REM REL3
2 REM LETTURA RECORD
100 DR$="0":NF$=DR$+":PROVA REL,L,":LU=21:CH=2:LF=2
110 BI=1:NR=30
120 OPEN15,8,15,"I"
130 OPENLF,8,CH,NF$+CHR$(LU)
140 GOSUB1000
145 PRINT"RECORD=0 PER USCIRE":PRINT
150 INPUT"RECORD ";R
155 IFR=0THEN290
160 IFR<0ORR>NRTHENPRINTCHR$(145);:GOTO150
170 HI=INT(R/256):LO=R-HI*256
180 C$="P"+CHR$(LF)
190 C$=C$+CHR$(LO)+CHR$(HI)
200 C$=C$+CHR$(BI)
240 PRINT#15,C$:GOSUB1000
250 INPUT#LF,N$,T$:GOSUB1000
260 PRINT"NOME: ";N$
270 PRINT"TEL.: ";T$
280 GOTO145
290 CLOSELF:GOSUB1000:CLOSE15:END
1000 INPUT#15,EN,EM$,ET,ES
1010 IFEN=0OREN=50THENRETURN
1020 PRINTEN;EM$;ET;ES
1030 STOP:RETURN
```

Questo programma legge il record voluto dal file e lo mostra sul video; il record viene letto partendo dalla prima posizione (BI=1).

```
1 REM REL4
2 REM LETTURA SOLO DEL SECONDO CAMPO
3 REM DI UN RECORD R
8 DR$="0":NF$=DR$+":PROVA REL,L,":LU=21:CH=2:LF=2
10 BI=12:NR=30
12 INPUT"RECORD ";R
```

```
15 IFR=0ORR>NRTHENPRINTCHR$(145);:GOTO10
20 OPEN15,8,15,"I"
30 OPENLF,8,CH,NF$+CHR$(LU)
40 GOSUB1000
50 HI=INT(R/256):LO=R-HI*256
60 C$="P"+CHR$(LF)
70 C$=C$+CHR$(LO)+CHR$(HI)
90 C$=C$+CHR$(BI)
100 PRINT#15,C$:GOSUB1000
120 INPUT#LF,T$:GOSUB1000
140 PRINT"TEL.: ";T$
150 CLOSELF:GOSUB1000:CLOSE15:END
1000 INPUT#15,EN,EM$,ET,ES
1010 IFEN=0OREN=50THENRETURN
1020 PRINTEN;EM$;ET;ES
1030 STOP:RETURN
```

Questo programma fa vedere come ponendo BI a 12 si legga il secondo campo di un record.

Non si consiglia di aprire contemporaneamente più di un file relativo.

Anche con i file relativi sussiste il problema del reperimento del record, se non risulta immediata l'associazione di un numero d'ordine univoco con un determinato record. Anche in questo caso si può procedere alla creazione di indici, primari o secondari, solo che, a differenza dei file random, basta memorizzare per ogni record il suo numero d'ordine nel file; esso costituisce l'indirizzo relativo del record (relativo rispetto all'inizio del file).

6.9 MESSAGGI DI ERRORE DEL DOS

Quando si ha un errore durante una operazione disco si vede lampeggiare l'indicatore di errore posto sull'unità. Lo stesso indicatore si accende durante una operazione disco, ma se non pulsa significa che tutto va bene. Nel caso si verifichi un errore, il sistema ripete un certo numero di volte le operazioni su disco prima di segnalare l'errore. L'indicatore di errore può essere interrogato da programma con le istruzioni che seguono, e questa operazione resetta (spegne) l'indicatore.

```
9000 INPUT#15,EN,EM$,ET,ES
9010 PRINT EN,EM$,ET,ES
```

Si suppone che sia già stata eseguita prima la: OPEN 15,8,15 per aprire il canale comandi. Si sono usati i numeri di linea 9000 e 9010; ognuno userà quello che crede, comunque è buona norma scrivere un sottoprogramma di errore e usarlo richiamandolo nei diversi punti del programma quando serve.

Le 4 variabili usate nel comando di INPUT possono avere nomi qualsiasi e possono essere variabili numeriche o stringa, salvo la seconda che deve riferirsi a una stringa, per contenere la descrizione del messaggio. Il contenuto delle variabili usate è ordinatamente:

EN codice del messaggio;
EM$ descrizione messaggio;
ET numero della traccia coinvolta;
ES numero del settore coinvolto.

Il messaggio di codice 0 significa OK, cioè tutto bene. Quello di codice 1 non segnala una condizione di errore, ma solo il numero dei file cancellati con una operazione di SCRATCH. I codici da 2 a 19 non sono usati dal sistema. Si riporta la descrizione e il significato degli altri messaggi.

20 READ ERROR
Il controllore del disco non riesce a trovare la testata (header) del settore richiesto; può essere invalido l'indirizzo passato o il disco è stato rovinato.

21 READ ERROR
Il controllore del disco non trova il carattere di sincronizzazione sulla traccia richiesta. Può essere allineata male la testina di lettura/scrittura, mancare il dischetto, essere introdotto male il dischetto, o non essere formattato, o infine può trattarsi di un guasto hardware.

22 READ ERROR
E' stato richiesto di leggere o verificare un blocco che non è stato scritto bene; con i comandi di tipo BLOCK segnala che si è dato un indirizzo illegale di traccia/settore.

23 READ ERROR
Nel blocco di dati letti il controllo CHECKSUM non dà risultato corretto; può anche indicare problemi di messa a terra.

24 READ ERROR
Nel blocco letto può essere errato un byte sia nella testata che nella parte dati; può anche indicare problemi di messa a terra.

25 WRITE ERROR
Indica un errore verificatosi nel controllo tra i dati scritti in memoria e quelli appena scritti sul disco.

26 WRITE PROTECT ON
Indica che si tenta di scrivere su un dischetto nel quale è stata chiusa la finestra di protezione contro la scrittura.

27 READ ERROR
Il controllore del disco ha trovato un errore di CHECKSUM nella testata del blocco e quindi non ha letto il blocco; può dipendere da problemi di messa a terra.

28 WRITE ERROR
Il controllore del disco cerca, dopo aver scritto un blocco, il carattere di sincronizzazione del blocco seguente e non lo trova (il carattere deve essere trovato entro un tempo definito, altrimenti si ha errore); può dipendere dal formato errato del dischetto o da un guasto hardware.

29 DISK ID MISMATCH
Indica il tentativo di accedere a un dischetto che non è stato inizializzato; può indicare anche la presenza di testate rovinate.

30 SYNTAX ERROR
Il DOS non riesce a interpretare il comando ricevuto; in generale dipende da errori nei parametri o nel modo di scriverli.

31 SYNTAX ERROR
Il DOS non riconosce il comando; può dipendere dal fatto di aver lasciato uno spazio prima del comando o di aver scritto un comando inesistente.

32 SYNTAX ERROR
Il comando inviato supera i 58 caratteri, lunghezza massima consentita.

33 SYNTAX ERROR
Il nome del file usato in comandi OPEN o SAVE non è valido.

34 SYNTAX ERROR
Il DOS non riconosce il nome del file; può essere stato scritto male o si sono dimenticati i due punti.

39 SYNTAX ERROR
Il comando inviato sul canale 15 non viene riconosciuto dal DOS.

50 RECORD NOT PRESENT
Si tenta di leggere dopo l'ultimo record con INPUT o GET. Tale errore si presenta in fase di preestensione o estensione dei file RELATIVI.

51 OVERFLOW IN RECORD
In una istruzione di PRINT si cerca di scrivere più caratteri di quelli contenuti in un blocco (settore); può succedere se nel computo dei caratteri non si considerano i caratteri separatori dei campi e il RETURN finale.

60 WRITE FILE OPEN
Indica che si apre per lettura un file che è stato scritto e non chiuso.

61 FILE NOT OPEN
Indica che si richiede un' operazione per un file che non è stato aperto; in alcuni casi non viene segnalato l'errore, ma l'operazione relativa al file viene ignorata.

62 FILE NOT FOUND
Si richiama un file che non esiste.

63 FILE EXISTS
Si cerca di creare un file il cui nome esiste già sul disco e non si usa il carattere "@" per cancellare il precedente.

64 FILE TYPE MISMATCH
Indica una incongruenza tra il tipo di file richiesto e quello presente sul disco con quel nome.

65 NO BLOCK
Indica che il blocco richiesto nell'operazione B-A è già occupato, ma fornisce nelle due variabili di traccia e settore il nuovo indirizzo disponibile; se non ci sono più blocchi liberi ritorna traccia e settore nulli.

66 ILLEGAL TRACK AND SECTOR
Il DOS tenta di accedere a una traccia e un settore inesistenti; può segnalare errori di formato nelle operazioni.

67 ILLEGAL SYSTEM T OR S
Indica una traccia e un settore illegali.

70 NO CHANNEL
Indica che il canale richiesto non è disponibile o che tutti i canali consentiti sono occupati. Si ricordi che possono essere aperti contemporaneamente 3 file sequenziali o 4 ad accesso diretto.

71 DIR ERROR
Indica che non tornano i controlli sulla BAM; si deve inizializzare di nuovo il dischetto, ma si danneggiano i file eventualmente aperti.

72 DISK FULL
Indica o che il disco è pieno o che la Directory non può registrate più entrate di file. Si ricordi che il numero massimo di entrate nella directory è 144.

73 CBM DOS 2.6 V170
Indica che si cerca di scrivere su un disco con formato non compatibile. Il DOS dell'unità 1540 è la versione 2.6, quello delle unità 2040 e 3040 è la versione 1.0, quello dell'unità 8050 è la versione 2.5, quello dell'unità 4040 è la versione 2.0. Si noti che per i formati dei dischi dei diversi sistemi CBM valgono le seguenti regole:
1.0 può essere letto ma non scritto dal VIC (2.6);
2.0 può essere letto e scritto dal VIC (2.6), si ha cioè completa compatibilità;
2.6 può essere letto, ma non scritto da 1.0;
2.6 può essere letto e scritto da 2.0;
2.6 non può nè essere letto nè scritto da 2.5.

74 DRIVE NOT READY
Indica che si tenta un'operazione disco, ma il dischetto non è in loco.

Si ricorda che per ottenere alcune di queste segnalazioni di errore si deve eseguire la routine di programma che legge le 4 variabili dal canale 15 (non sempre il sistema segnala spontaneamente gli errori). Queste variabili possono essere evidenziate sul video e/o il programma deve scegliere, di conseguenza, la strada giusta, che potrebbe anche essere lo STOP. Se non si interroga il canale 15 capiterà di vedere pulsare la luce di errore e non si saprà da cosa dipende. In tale caso non deve sembrare strano avere rovinato i propri dischetti

6.10 PROGRAMMI DI UTILITA'

Si chiamano programmi di utilità quei programmi che aiutano a gestire le apparecchiature. Per l'unità 1540 viene fornito un dischetto (TEST/DEMO) che contiene alcuni programmi di questo tipo. In questo paragrafo si commentano alcuni tra i programmi dimostrativi e se ne aggiunge qualche altro.

Dopo aver montato il dischetto TEST/DEMO si scrive:

OPEN 15,8,18,"I"
LOAD "$"
LIST

Si vedrà la seguente lista della directory:

```
0  1540TEST/DEMO [illegible]
4     "DIR"                PRG
6     "VIEW BAM"           PRG
14    "DISPLAY T&S"        PRG
4     "CHECK DISK"         PRG
9     "PERFORMANCE TEST"   PRG
5     "SEQUENTIAL FILE"    PRG
13    "RANDOM FILE"        PRG
609 BLOCKS FREE.
```

Nel commentare i programmi di utilità del dischetto non si riportano i listati, questi si possono facilmente ottenere, dopo averli caricati in memoria, con il comando LIST.

Il programma PERFORMANCE TEST serve per provare il buon funzionamento del dischetto. Dopo averlo caricato e dato il RUN si vedono sul video una serie di messaggi; se tutto va bene i messaggi sono i seguenti:

PERFORMANCE TEST	Traduzione
INSERT SCRATCH DISKETTE IN DRIVE PRESS RETURN WHEN READY WAIT ABOUT 80 SECONDS	Inserisci un dischetto cancellabile premi RETURN quando sei pronto attendi 80 secondi circa

A questo punto si deve eseguire il test, ma si deve stare attenti e montare un dischetto o nuovo o contenente dati che non servono più, perchè esso viene cancellato. Il dischetto riceve il nome di "Test Disk" e viene formattato. I messaggi continuano così:

0 OK 0 0	cont. 4 var. ottenute con INPUT#15
DRIVE PASS	controllo meccanico
MECANICAL TEST	dischetto
OPEN WRITE FILE	apertura file per scrivere
0 OK 0 0	variabili controllo
WRITING DATA	scrittura dati
0 OK 0 0	variabili di controllo
CLOSE WRITE DATA	chiusura file scritto
0 OK 0 0	variabili di controllo
OPEN READ FILE	apertura file per leggere
0 OK 0 0	variabili di controllo
READING DATA	lettura dato
0 OK 0 0	variabili di controllo
SCRATCH FILE	cancella file scritto
1 FILE SCRATCHED 1 0	variabili di controllo che indicano la cancellazione (cod.1) alla traccia 1, settore 0
WRITE TRACK 35	scrive in modo diretto nella traccia 35
0 OK 0 0	variabili di controllo
WRITE TRACK 1	scrive nella traccia 1

0 OK 0 0	variabili di controllo
READ TRACK	35 legge la traccia 35
0 OK 0 0	variabili di controllo
READ TRACK 1	legge la traccia 1
0 OK 0 0	variabili di controllo
UNIT HAS PASSED	l'unità ha superato
PERFORMANCE TEST!	il test
PULL DISKETTE FROM	togli il dischetto
DRIVE BEFORE TURNING	prima di spegnere
POWER OFF.	il calcolatore.

Se la messaggistica che appare sul video è diversa da quella sopra riportata, significa che qualcosa è andato male e conviene riprovare; se gli errori persistono potrebbe essere irrimediabilmente danneggiato il dischetto e conviene provare con un dischetto nuovo. Se anche così le cose non funzionano conviene rivolgersi al rivenditore per far controllare l'unità disco o l'interfaccia.

Per capire come è fatto il programma si può listarlo o sul video a pezzi o sulla stampante. La cosa può essere interessante dato che compaiono parecchi dei comandi prima passati in rassegna. Si raccomanda di guardare la routine di errore da 1840 a 1900; essa segnala errore solo se il codice dell'errore è maggiore o uguale a 2.

Il programma DIR presenta un certo interesse. Esso dopo il RUN mostra il seguente MENU sul video:

D-DIRECTORY
>-DISK COMMAND
Q-QUIT PROGRAM
S-DISK STATUS

e si può scegliere una delle 3 funzioni, oppure uscire dal programma con Q. All'inizio il canale comandi è stato aperto con il numero logico di file uguale a 2. Se si risponde con D il programma va alla linea 10 e qui viene aperta la directory come file "$0" sul canale 0 e con numero logico di file uguale a 1, dopo con delle GET#1 viene letta la directory carattere per carattere e stampata. Si ottiene quindi la directory con una procedura diversa che con i comandi LOAD "$" e LIST.
Se si risponde con S il programma va a leggere, ancora in un modo diverso dal solito, lo stato del programma riguardo alle operazioni disco. Infatti sul canale comandi, aperto con il numero logico 2, vengono operate una serie di GET# fino a quando non si incontra il RETURN, stampando carattere per carattere quello che si legge. Il risultato è il medesimo di quando si opera con la solita routine di errore e si usa il comando INPUT# seguito dai nomi delle 4 variabili. Queste istruzioni si trovano da 5000 a 5020.

Se si risponde con ">" il programma stampa il carattere e si mette in attesa di ricevere una stringa di comandi disco; quando la stringa è terminata, cioè si preme RETURN, il programma esegue il comando con una PRINT#2 e subito dopo stampa il messaggio di eventuale errore.

Segue il listato del programma DIRMODIFICATO, ottenuto appportando alcune moâifiche al programma DIR. Le modifiche sono le seguenti:

.aggiunta della linea 15 per il controllo dei blocchi occupati con la variabile BO;
.aggiunta della linea 75 per la somma in BO dei blocchi occupati;
.aggiunta delle linee da 1002 a 1008, nelle quali viene emesso un messaggio di segnalazione se il totale blocchi non è 664, accompagnandolo con un segnale sonoro;
.modifica delle linee 4012 e 4013 e aggiunta delle linee 4014 e 4015 per migliorare la ricezione del comando disco da tastiera: vengono accettati tutti i caratteri tra spazio e z, viene accettato il carattere DEL (ASCII=20) e corretta la stringa.

```
1 REM DIR MODIFICATO
4 OPEN2,8,15
5 PRINT"":GOTO 10000
10 OPEN1,8,0,"$0"
15 BO=0
20 GET#1,A$,B$
30 GET#1,A$,B$
40 GET#1,A$,B$
50 C=0
60 IF A$<>"" THEN C=ASC(A$)
70 IF B$<>"" THEN C=C+ASC(B$)*256
75 BO=BO+C
80 PRINT""MID$(STR$(C),2);TAB(3);"";
90 GET#1,B$:IF ST<>0 THEN 1000
100 IF B$<>CHR$(34) THEN 90
110 GET#1,B$:IF B$<>CHR$(34)THEN PRINTB$;:GOTO110
120 GET#1,B$:IF B$=CHR$(32) THEN 120
130 PRINT TAB(18);:C$=""
140 C$=C$+B$:GET#1,B$:IF B$<>""  THEN 140
150 PRINT""LEFT$(C$,3)
160 GET T$:IF T$<>"" THEN GOSUB 2000
170 IF ST=0 THEN 30
1000 PRINT"BLOCCHI LIBERI"
1002 IFBO=664THEN1010
1003 PRINT
1005 FORC=1TO20:PRINT"RISULTATO TEST BLOCCHI"SPC(9)BO-664:POKE36875,220
1006 POKE36878,15:FORI=1TO500:NEXT:POKE36878,0:IFC=20THEN1008
1007 PRINT"                                  ":PRINT"";
1008 FORI=1TO200:NEXT:NEXT:POKE36875,0
1010 CLOSE1:GOTO 10000
2000 IF T$="Q" THEN CLOSE1:END
2010 GET T$:IF T$="" THEN 2000
2020 RETURN
4000 REM DISK COMMAND
4010 C$="":PRINT">";
4011 GETB$:IFB$="" THEN4011
4012 IF B$=CHR$(13) THEN 4020
4013 IFB$>=" "ANDB$=<"Z"THENPRINTB$;:C$=C$+B$:GOTO4011
```

```
4014 IFASC(B$)=20ANDC$=""THENPRINTB$:GOTO10000
4015 IFASC(B$)=20THENPRINTB$;:C$=LEFT$(C$,LEN(C$)-1)
4016 GOTO4011
4020 PRINT:PRINT#2,C$
5000 PRINT"■";
5010 GET#2,A$:PRINTA$;:IF A$<>CHR$(13)GOTO5010
5020 PRINT"■"
10000 PRINT "D-DIRECTORY"
10010 PRINT ">-COMANDI DISCO"
10020 PRINT "Q-USCITA PROGRAMMA"
10030 PRINT "S-STATO DISCO"
10100 GETA$:IFA$=""THEN10100
10200 IF A$="D" THEN 10
10300 IF A$="." OR A$=">" OR A$=">" THEN 4000
10310 IF A$="Q" THEN END
10320 IF A$="S" THEN 5000
10999 GOTO 10100
```

Si riporta il listato del programma CONTR/DISCO che serve per controllare se i file di tipo 1 (SEQ) e di tipo 2 (PRG) sono registrati bene.

```
1 REM CONTR/DISCO
10 OPEN1,8,15,"I":OPEN2,8,2,"#":OPEN3,8,3,"#"
20 DEFFNF(X)=2+32*X
30 TD=18:SD=1:GOSUB9000
40 FI$="":CB=0
45 PRINT#1,"B-P:2,"FNF(FI)
50 GET#2,A$:GOSUB10010:TY=ASC(A$)-128
60 GET#2,A$,B$:GOSUB10000:TR=ASC(A$):SC=ASC(B$)
70 FORI=1TO16:GET#2,A$:FI$=FI$+A$:IFA$=CHR$(160)THENFI$=LEFT$(FI$,I-1):I=16
80 NEXT
90 PRINT#1,"B-P:2,"FNF(FI)+28:GET#2,A$,B$:GOSUB10000:NB=ASC(A$)+ASC(B$)*256
95 IFTY<>2ANDTY<>1THENFI=FI+1:IFFI=8THENGOSUB9000:FI=0
97 IFTY<>2ANDTY<>1THEN40
100 PRINT"■■■■*************************■■■■   CONTROLLO   FILE■■■"
105 PRINTTAB((21-LEN(FI$))/2)FI$"■■■"
110 PRINT#1,"B-R:3,0,"TR,SC
120 PRINT#1,"B-P:3,0"
130 CB=CB+1:GET#3,A$,B$:GOSUB10000:TR=ASC(A$):SC=ASC(B$):PRINT"●";
140 IFTR<>0THEN110
150 IFCB=NBTHENPRINT:PRINTTAB(9)"■■■■■OK■":FI=FI+1:IFFI=8THENGOSUB9000:FI=0
160 IFCB=NBTHENFORI=1TO250:NEXTI:GOTO40
170 PRINT:PRINT"■FILE NON CORRETTO     :■■"TAB((21-LEN(FI$))/2)FI$"■"
180 PRINT"■DEVO AZZERARE  ? (■S■/■N■)"
200 GETA$:IFA$=""THEN200
205 IFA$="N"THEN11000
210 IFA$<>"S"THEN200
215 PRINT"■AZZERAMENTO E VALIDATE"
220 PRINT#1,"S:"FI$:PRINT#1,"V":CLOSE3:CLOSE2:CLOSE1:RUN
9000 IFTD=0THENPRINT"■TUTTO IN ORDINE    ...■ SPERO !":GOTO11000
9005 A$="":B$="":PRINT#1,"U1:2,0";TD;SD
9010 GET#2,A$,B$:GOSUB10000:TD=ASC(A$):SD=ASC(B$):RETURN
10000 IFB$=""THENB$=CHR$(0)
10010 IFA$=""THENA$=CHR$(0)
10020 RETURN
11000 CLOSE3:CLOSE2:CLOSE1:STOP
```

In questo programma viene aperto il canale 15 come file logico 1 e vengono aperti 2 canali per accesso diretto, il 2 e il 3, come file logici 2 e 3 rispettivamente. Il canale 2 viene usato per leggere la directory; per ogni file di tipo 1 o 2 viene ricercato il nome, l'indirizzo di inizio e il numero dei blocchi. Dopo viene letto il file usando il canale 3 e sfruttando i primi 2 byte per il concatenamento dei settori; viene stampato un pallino per ogni blocco controllato. Se il conto dei blocchi torna, tutto va bene, altrimenti viene chiesto se si vuole azzerare il file e in caso di risposta affermativa viene eseguita questa operazione; poi si esegue la VALIDATE del disco e viene rifatto il controllo.

Il programma VIEW BAM mostra sul video la BAM del dischetto; si vedono dapprima le tracce da 1 a 17 e poi le tracce da 18 a 35. La BAM è vista come una matrice, sull'asse orizzontale sono riportate le tracce e sull'asse verticale i settori; questi ultimi vanno da 0 a 20 anche se su alcuni gruppi di tracce ne sono presenti di meno. Nella matrice mostrata le posizioni scure indicano i blocchi occupati. Si vede che è in parte occupata la traccia 18, quella della BAM e della directory, e che i diversi file sono allocati a partire dalle tracce vicine alla 18. Nel programma si ha una imperfezione e per le tracce da 18 a 24 viene mostrato come occupato il settore 19 che in queste tracce non esiste.

Si riporta il listato del programma BAMSUPRINTER, che è ottenuto dal precedente aggiungendo, dalla linea 60000 alla linea 60140, un sottoprogramma per ottenere l'hard copy del video e le linee 261 e 321 che richiamano questo sottoprogramma per avere la stampa del contenuto del video. Si riportano anche le 2 stampe: escludendo il settore 19, dove non deve essere presente, si contano 76 blocchi occupati, si tratta di un dischetto dove i file registrati occupano 73 settori; gli altri 3 sono occupati da BAM e directory. Si noti che la traccia 18 non risulta occupata tutta, anche se non compare come libera nel computo dei settori liberi (i 664 iniziali dopo la formattazione). Il sottoprogramma funziona con configurazione del VIC standard, con la mappa video che inizia a 7680 (7658=7680-22); volendo si può cambiare il valore della costante G1 alla linea 60020 per poterlo usare anche con altre configurazioni.

```
1 REM BAMSUPRINTER
105 OPEN15,8,15
110 PRINT#15,"I0":NU$="N/A N/A N/A N/A N/A":Z4=1
120 OPEN2,8,2,"#"
130 Y$="[illegible]"
140 X$="[illegible]"
150 DEF FNS(Z) = 2↑(S-INT(S/8)*8) AND (SB(INT(S/8)))
160 PRINT#15,"U1:";2;0;18;0
170 PRINT#15,"B-P";2;1
180 PRINT"[illegible]";
190 Y=22:X=1:GOSUB430
200 FORI=0TO20:PRINT:PRINT"[illegible]"RIGHT$(STR$(I)+" ",3);:NEXT
210 GET#2,A$
220 GET#2,A$
230 GET#2,A$
240 TS=0
250 FORT=1TO17:GOSUB450
```

```
260 Y=22:X=T+4:GOSUB430:GOSUB540:NEXT
261 GOSUB60000
270 FORI=1TO2000:NEXT:PRINT"]"
280 Y=22:X=1:GOSUB430
290 FORI=0TO20:PRINT:PRINT"]]"RIGHT$(STR$(I)+"  ",3);:NEXT
300 FORT=18TO35
310 GOSUB450
320 Y=22:X=T-13:GOSUB430:GOSUB540:NEXT
321 GOSUB60000
330 FORI=1TO1000:NEXT
340 PRINT"]"
350 PRINT#15,"B-P";2;144
360 N$="":FORI=1TO20:GET#2,A$:N$=N$+A$:NEXT
370 PRINT" "N$" "TS-17;"BLOCCHI LIBERI"
380 FORI=1TO4000:NEXT
390 PRINT"]"
400 INPUT"ALTRO DISCHETTO  N";A$
410 IFA$="S"THENRUN
420 IFA$<>"S"THENEND
430 PRINTLEFT$(Y$,Y)LEFT$(X$,X)"";
440 RETURN
450 GET#2,SC$:SC=ASC(RIGHT$(CHR$(0)+SC$,1))
460 TS=TS+SC
470 GET#2,A$:IFA$=""THENA$=CHR$(0)
480 SB(0)=ASC(A$)
490 GET#2,A$:IFA$=""THENA$=CHR$(0)
500 SB(1)=ASC(A$)
510 GET#2,A$:IFA$=""THENA$=CHR$(0)
520 SB(2)=ASC(A$)
530 RETURN
540 PRINT""RIGHT$(STR$(T),1);"";
550 REM  PRINTT"   "SC"  "SB(0)" "SB(1)" "SB(2)=CHR$(0)
560 IFT>24ANDS=18THEN:PRINTMID$(NU$,Z4,1);:GOTO660
570 FORS=0TO20
580 IFT<18THEN620
590 IFT>30ANDS=17THEN:PRINTMID$(NU$,Z4,1);:GOTO660
600 IFT>24ANDS=18THEN:PRINTMID$(NU$,Z4,1);:GOTO660
610 IFT>24ANDS=19THENPRINTMID$(NU$,Z4,1);:GOTO660
620 IFT>17ANDS=20THENPRINTMID$(NU$,Z4,1);:Z4=Z4+1:GOTO660
630 PRINT"";
640 IF FNS(S)=0 THEN PRINT"+";:GOTO660
650 PRINT"+";
660 PRINT"";
670 NEXT
680 RETURN
60000 REM HARD COPY
60010 G1$=CHR$(145)
60020 OPEN4,4:PRINT#4:G1=7658
60030 FORG0=0TO22:G0$=G1$:G1=G1+22
60040 FORG2=G1TOG1+21:G3=PEEK(G2)
60050 IFG3>128THENG3=G3-128:G4=1:G0$=G0$+CHR$(18)
60060 IF(G3>0)*(G3<32)THENG3=G3+64:GOTO60100
60070 IF(G3>31)*(G3<64)THEN60100
60080 IF(G3>63)*(G3<96)THENG3=G3+128:GOTO60100
60090 IF(G3>95)*(G3<128)THENG3=G3+64:GOTO60100
60100 G0$=G0$+CHR$(G3)
60110 IFG4=1THENG0$=G0$+CHR$(146):G4=0
60120 NEXTG2:PRINT#4,G0$:NEXTG0
60130 PRINT#4:CLOSE4
60140 RETURN
```

Listato prodotto dal programma precedente:

```
20 ++++++++++++++■■■          20 N/A N/A N/A N/A N/
19 ++++++++++++++■■■          19 ■■■■■■■ N/A N/A N/
18 ++++++++++++++■■■          18 +■■++++ N/A N/A N/
17 ++++++++++++++■■■          17 +■■+++++++++/A N/
16 ++++++++++++++■■■          16 +■■+++++++++++++++
15 ++++++++++++++■■■          15 +■+++++++++++++++
14 ++++++++++++++■■■          14 +■+++++++++++++++
13 ++++++++++++++■■■          13 +■+++++++++++++++
12 ++++++++++++++■■■          12 +■+++++++++++++++
11 ++++++++++++++■■■          11 +■■+++++++++++++++
10 ++++++++++++++■■■          10 +■■+++++++++++++++
 9 ++++++++++++++■■■           9 +■■+++++++++++++++
 8 ++++++++++++++■■■           8 +■■+++++++++++++++
 7 ++++++++++++++■■■           7 +■■+++++++++++++++
 6 ++++++++++++++■■■           6 +■■+++++++++++++++
 5 ++++++++++++++■■■           5 +■+++++++++++++++
 4 ++++++++++++++■■■           4 ■■+++++++++++++++
 3 ++++++++++++++■■■           3 +■+++++++++++++++
 2 ++++++++++++++■■■           2 +■■+++++++++++++++
 1 ++++++++++++++■■■           1 ■■■+++++++++++++++
 0 ++++++++++++++■■■           0 ■■■+++++++++++++++

   12345678901234567             890123456789012345
```

Il programma DISPLAY T&S serve per vedere o sul video o sulla stampante il contenuto dei settori del disco; vengono mostrati i contenuti dei 256 byte in esadecimale a sinistra, mentre a destra sono stampati i caratteri corrispondenti. Si riporta il risultato su stampante del contenuto del settore 1 della traccia 18 del dischetto TEST/DEMO, al quale sono stati aggiunti alcuni programmi.

```
TRACK 18  SECTOR 1

00 :12 04 82 11 00 44 49 52 A0 A0 A0 A0 A0 A0 A0 A0 :  ▮ DIR
10 :A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 04 00 :
20 :00 00 82 11 01 56 49 45 57 20 42 41 4D A0 A0 A0 :  ▮ VIEW BAM
30 :A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 06 00 :
40 :00 00 82 11 04 44 49 53 50 4C 41 59 20 54 26 53 :  ▮ DISPLAY T&S
50 :A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 0E 00 :
60 :00 00 82 13 00 43 48 45 43 4B 20 44 49 53 4B A0 :  ▮ CHECK DISK
70 :A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 04 00 :
80 :00 00 82 13 02 50 45 52 46 4F 52 4D 41 4E 43 45 :  ▮ PERFORMANCE
90 :20 54 45 53 54 00 00 00 00 00 00 00 00 00 09 00 : TEST
A0 :00 00 82 13 07 53 45 51 55 45 4E 54 49 41 4C 20 :  ▮ SEQUENTIAL
B0 :46 49 4C 45 A0 00 00 00 00 00 00 00 00 00 05 00 :FILE
C0 :00 00 82 13 10 52 41 4E 44 4F 4D 20 46 49 4C 45 :  ▮ RANDOM FILE
D0 :A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 0D 00 :
E0 :00 00 82 10 00 48 41 52 44 43 4F 50 59 A0 A0 A0 :  ▮ HARDCOPY
F0 :A0 A0 A0 A0 A0 00 00 00 00 00 00 00 00 00 02 00 :
```

Si vede che i 2 primi byte contengono in decimale 18 e 4, cioè questo settore è concatenato al 4 della stessa traccia, dove continua la directory. Si vede che le

entrate dei file sono separate da 2 byte contenenti zero binario. Il primo file registrato è di tipo PRG, infatti il terzo byte (primo dell'entrata) contiene in decimale 130; togliendo 128, si ha 2, tipo del file PRG. Seguono due byte con indirizzo di traccia e settore dell'inizio del programma e si trova: t=17, s=0. Seguono 16 byte con il nome del programma completato con il codice dello spazio+shift; si trova DIR (in decimale i codici ASCII 68, 73 e 82 e poi 13 volte 160). I byte seguenti sono a zero binario e alla fine gli ultimi 2 byte dell'entrata (il 29esimo e il 30esimo) contengono il numero dei settori occupati, che qui è 4.
Se si stampano i contenuti del settore 0 della traccia 17 si vede, molto male, il programma; infatti molti byte non corrispondono a caratteri stampabili. Si ricordi che i primi due byte del settore sono usati per il concatenamento; in questo esempio i 4 blocchi del programma si trovano in: t=17/s=0, t=17/s=10, t=17/s=20, t=17/s=8. Si scopre così che il sistema non registra i programmi in settori consecutivi.
Segue il listato del programma TRAC/SET, che è un programma ricavato dal precedente DISPLAY T&S apportando alcune modifiche. Sono state abolite molte linee che con delle REM miglioravano la impaginazione del programma, sono stati tradotti in italiano i messaggi, è stato modificato il modo di leggere il byte di posizione zero del buffer (linee da 350 a 420); dopo ogni lettura e stampa di un blocco vengono chiusi tutti i canali (routine 1000) e poi viene reinizializzato il disco, con riapertura dei canali (routine 1050). Con questo programma si può stampare blocco dopo blocco qualunque file che rechi nei primi 2 byte il concatenamento alla traccia e settore del blocco logicamente seguente, oppure stampare blocchi singoli senza concatenamento.

```
  REM TRAC/SET
150 PRINT"________________________";
160 PRINT"  CONTENUTO BLOCCHI   ";
165 PRINT"________________________";
170 REM COSTANTI
200 SP$=" ":NL$=CHR$(0):HX$="0123456789ABCDEF"
210 FS$="":FORI=64 TO 95:FS$=FS$+""+CHR$(I)+"":NEXT I
220 SS$="  ":FOR I=192 TO 223:SS$=SS$+""+CHR$(I)+"":NEXT I
240 DIM A$(15),NB(2)
251 D$="0"
253 PRINT"          VIDEO D  PRINTER"
254 GETJJ$:IF JJ$="" THEN254
255 IF JJ$="V"THENPRINT"         VIDEO "
256 IF JJ$="P"THENPRINT"         PRINTER"
265 OPEN4,4
270 GOSUB1050
280 REM CARICA BUFFER
320 INPUT"TRACK, SECTOR";T,S
330 IF T=0 OR T>35 THEN PRINT#15,"I"D$:GOSUB1000:CLOSE4:PRINT"END":END
340 IF JJ$="V" THEN PRINT"TRACK"T" SECTOR"S""
341 IF JJ$="P" THEN PRINT#4:PRINT#4,"TRACCIA"T" SETTORE"S:PRINT#4
350 PRINT#15,"U1:2,"D$;T;S:GOSUB630
355 PRINT#15,"B-P:2,0"
360 REM LEGGE BYTE 0
```

```
420 GET#2,A$(0):IFA$(0)=""THENA$(0)=NL$
425 PRINT#15,"B-P:2,1"
428 IF JJ$="V"THEN430
430 IF JJ$="P"THEN460
431 REM VIDEO
436 K=1:NB(1)=ASC(A$(0))
438 FOR J=0 TO 63:IF J=32 THEN GOSUB 710:IF Z$="N"THEN J=80:GOTO 458
440 FOR I=K TO 3
442 GET#2,A$(I):IF A$(I)="" THEN A$(I)=NL$
444 IF K=1 AND I<2 THEN NB(2)=ASC(A$(I))
446 NEXT I:K=0
448 A$="":B$=":":N=J*4:GOSUB 790:A$=A$+":"
450 FOR I=0 TO 3:N=ASC(A$(I)):GOSUB 790
452 C$=A$(I):GOSUB 850:B$=B$+C$
454 NEXT I:IF JJ$="V" THEN PRINTA$B$
458 NEXT J:GOTO571
460 REM PRINTER
466 K=1:NB(1)=ASC(A$(0))
468 FOR J=0 TO 15
470 FOR I=K TO 15
472 GET#2,A$(I):IF A$(I)="" THEN A$(I)=NL$
474 IF K=1 AND I<2 THEN NB(2)=ASC(A$(I))
476 NEXT I:K=0
478 A$="":B$=":":N=J*16:GOSUB 790:A$=A$+":"
480 FOR I=0 TO 15:N=ASC(A$(I)):GOSUB 790
482 C$=A$(I):GOSUB 850:B$=B$+C$
484 NEXT I
486 IF JJ$="P" THEN PRINT#4,A$B$
488 NEXT J:GOTO571
571 REM SUCCESSIVO BLOCCO
575 PRINT"TRACCIA/SETT.SEGUENTE"NB(1)NB(2) "■"
580 PRINT"DESIDERI PROSEGUIRE         S/N   "
590 GET Z$:IF Z$="" THEN590
600 IF Z$="S" THEN T=NB(1):S=NB(2):GOSUB1000:GOSUB1050:GOTO330
610 IF Z$="N" THEN GOSUB1000:GOSUB1050:GOTO320
620 GOTO 590
630 REM ROUTINE ERRORE
680 INPUT#15,EN,EM$,ET,ES:IF EN=0 THEN RETURN
690 PRINT"■DISK ERROR■"EN,EM$,ET,ES
700 END
710 REM MESS. CONTINUAZIONE
740 PRINT"■■■■■■CONTINUO(S/N)■"
750 GETZ$:IF Z$="" THEN 750
760 IF Z$="N" THEN RETURN
770 IF Z$<>"S" THEN 750
780 PRINT"■TRACCIA" T " SETTORE"S "■":RETURN
790 REM CONV.HEX
820 A1=INT(N/16):A$=A$+MID$(HX$,A1+1,1)
830 A2=INT(N-16*A1):A$=A$+MID$(HX$,A2+1,1)
840 A$=A$+SP$:RETURN
850 REMCONV.ASCII
890 IF ASC(C$)<32 THEN C$=" ":RETURN
910 IF ASC(C$)<128 OR ASC(C$)>159 THEN RETURN
920 C$=MID$(SS$,3*(ASC(C$)-127),3):RETURN
1000 CLOSE2:CLOSE15:RETURN
1050 OPEN15,8,15,"I"+D$:GOSUB630:OPEN2,8,2,"#":GOSUB630:RETURN
```

Il programma CHECK DISK serve per controllare se un disco reca settori dove non si può scrivere. All'inizio il programma fa la VALIDATE del disco e quindi cancella settori occupati, ma non segnalati nella Directory. Poi prova a scrivere nei settori rimasti liberi, occupandoli anche nella BAM, ma non nella Directory; se trova settori nei quali non si può scrivere li annota in una tabella come settori guasti. Alla fine segnala se ha trovato settori non scrivibili. Si deve fare attenzione al fatto che per poter utilizzare il disco bisognerebbe rifare la VALIDATE per liberare i settori non segnalati nella directory, poi allocare i settori guasti (come se fossero occupati) senza andarci a scrivere. Il programma in realtà lascia il disco tutto occupato; sia i settori buoni che quelli non buoni.

Segue il programma SIST/DISCO ricavato dal precedente con alcune modifiche. Esso lascia il disco in stato da poter essere usato, ma con allocati (occupati) i settori rovinati. Si ricordi di non fare la VALIDATE su un disco che ha queste caratteristiche.

```
1 REM SIST/DISCO
5 DIMT(100):DIMS(100):REM TRACCE E SETTORI GUASTI
9 PRINT"▮▮▮▮▮────────────────────────";
10 PRINT"      SIST/DISCO      "
12 PRINT"──────────────────────────────"
30 OPEN15,8,15
35 PRINT#15,"V0"
45 N%=RND(TI)*255
50 A$="":FORI=1TO255:A$=A$+CHR$(255AND(I+N%)):NEXT
60 GOSUB900
70 OPEN2,8,2,"#"
80 PRINT:PRINT#2,A$;
85 T=1:S=0
90 PRINT#15,"B-A:0"T;S
100 INPUT#15,EN,EM$,ET,ES
110 IFEN=0THEN130
115 IFET=0THEN200:REM END
120 PRINT#15,"B-A:0"ET;ES:T=ET:S=ES
130 PRINT#15,"U2:2,0"T;S
134 NB=NB+1:PRINT"PROVATI BLOCCHI "NB
135 PRINT"TRACCIA    ▮▮▮▮▮"T;" SETTORE   ▮▮▮▮"S
140 INPUT#15,EN,EM$,ES,ET
150 IF EN=0THEN85
160 T(J)=T:S(J)=S:J=J+1
165 PRINT"▮▮▮▮BLOCCHI GUASTI:";T;S
170 GOTO85
200 PRINT#15,"I0"
210 GOSUB900
212 CLOSE2
215 IFJ=0THENPRINT"▮▮▮▮▮▮TUTTO BENE!   ":CLOSE15:END
217 OPEN2,8,2,"#"
218 PRINT"▮▮▮▮BLOCCHI GUASTI","TRACCIA","SETTORE▮"
219 PRINT#15,"V0":PRINT#15,"I0"
220 FORI=0TOJ-1
230 PRINT#15,"B-A:0"T(I);S(I)
240 PRINT,,T(I),S(I)
250 NEXT
260 PRINT"▮"J" BLOCCHI GUASTI    SONO STATI ALLOCATI"
```

```
270 CLOSE2:CLOSE15:END
900 INPUT#15,EN,EM$,ET,ES
910 IF EN=0 THEN RETURN
920 PRINT"[illegible]ERRORE#"EN,EM$;ET;ES"[illegible]"
930 PRINT#15,"I0"
940 RETURN
```

Segue il programma NUMBUFFER, che serve per vedere quanti file di tipo RANDOM possono essere aperti contemporaneamente. Provandolo si vede che al massimo se ne possono aprire 4; se si tenta di aprirne di più, si ha la segnalazione di NO CHANNEL. Il programma mostra anche il numero del buffer, situato nella RAM del disco, che viene assegnato quando si apre un file. L'operazione di GET sul canale va fatta subito dopo l'apertura di questo e il carattere letto è una stringa il cui codice ASCII è il numero del buffer.

```
1 REM NUMBUFFER
10 OPEN15,8,15,"I"
15 INPUT"QUANTI BUFFER";N
17 A$="":B$="":C$="":D$="":E$=""
19 ONNGOTO60,50,40,30,20
20 OPEN2,8,2,"#":GOSUB100:GET#2,E$:GOSUB100
30 OPEN3,8,3,"#":GOSUB100:GET#3,D$:GOSUB100
40 OPEN4,8,4,"#":GOSUB100:GET#4,C$:GOSUB100
50 OPEN5,8,5,"#":GOSUB100:GET#5,B$:GOSUB100
60 OPEN6,8,6,"#":GOSUB100:GET#6,A$:GOSUB100
65 ONNGOTO78,76,74,72,70
70 IFE$=""THENE$=CHR$(0)
71 PRINTASC(E$):CLOSE2
72 IFD$=""THEND$=CHR$(0)
73 PRINTASC(D$):CLOSE3
74 IFC$=""THENC$=CHR$(0)
75 PRINTASC(C$):CLOSE4
76 IFB$=""THENB$=CHR$(0)
77 PRINTASC(B$):CLOSE5
78 IFA$=""THENA$=CHR$(0)
79 PRINTASC(A$):CLOSE6
80 CLOSE15:STOP
100 INPUT#15,EN,EM$,ET,ES
110 PRINTEN;EM$;ET;ES
120 RETURN
```

Per provare il programma è meglio, prima di lanciarlo, spegnere e poi riattivare l'unità disco, così si azzera la RAM del disco. Si possono fare diverse prove, e si vede come il sistema assegna i buffer al programma.

Segue il programma INDBUFFER, il cui scopo è quello di mostrare come il sistema assegna i buffer al programma e quale è il loro indirizzo.

```
1 REM INDBUFFER
5 DIMR$(8,7):FORK=1TO8:FORI=1TO7:R$(K,I)="":NEXTI:NEXTK
7 OPEN7,4:PRINT#7,"RISULTATI INDBUFFER"
10 OPEN15,8,15,"I"
15 INPUT"QUANTI BUFFER";N:PRINT#7:PRINT#7,N;" BUFFER":PRINT#7
17 A$="":B$="":C$="":D$="":E$=""
19 ONNGOTO60,50,40,30,20
20 OPEN2,8,2,"#":GOSUB997:GET#2,E$:GOSUB997
30 OPEN3,8,3,"#":GOSUB997:GET#3,D$:GOSUB997
40 OPEN4,8,4,"#":GOSUB997:GET#4,C$:GOSUB997
50 OPEN5,8,5,"#":GOSUB997:GET#5,B$:GOSUB997
60 OPEN6,8,6,"#":GOSUB997:GET#6,A$:GOSUB997
65 ONNGOTO450,400,350,300,250
250 IFE$=""THENE$=CHR$(0)
255 PRINT#7,ASC(E$)
260 PRINT#15,"B-P:"2;1
265 PRINT#2,"22222";:GOSUB997
300 IFD$=""THEND$=CHR$(0)
305 PRINT#7,ASC(D$)
310 PRINT#15,"B-P:"3;1
315 PRINT#3,"33333";:GOSUB997
350 IFC$=""THENC$=CHR$(0)
355 PRINT#7,ASC(C$)
360 PRINT#15,"B-P:"4;1
365 PRINT#4,"44444";:GOSUB997
400 IFB$=""THENB$=CHR$(0)
405 PRINT#7,ASC(B$)
410 PRINT#15,"B-P:"5;1
415 PRINT#5,"55555";:GOSUB997
450 IFA$=""THENA$=CHR$(0)
455 PRINT#7,ASC(A$)
460 PRINT#15,"B-P:"6;1
465 PRINT#6,"66666";:GOSUB997
500 FORK=1TO8:M=K-1
510 FORI=1TO7
520 L=I-1
530 PRINT#15,"M-R"CHR$(L)CHR$(M):GET#15,R$(K,I)
540 NEXTI
550 NEXTK
989 FORK=1TO8:FORI=1TO7:IFR$(K,I)=""THENR$(K,I)=CHR$(0)
991 NEXTI:NEXTK
993 FORK=1TO8:FORI=1TO7:PRINT#7,K,I,R$(K,I):NEXTI:NEXTK
995 FORK=1TO7:CLOSEK:NEXTK:CLOSE15:STOP
997 INPUT#15,EN,EM$,ET,ES
998 PRINTEN;EM$;ET;ES
999 RETURN
```

Inizialmente il programma chiede quanti buffer (file) si vogliono, e si è già visto che il numero massimo consentito è 4. Dopo la risposta, il programma apre il numero richiesto di file e per ognuno legge con la GET# il numero del buffer assegnato e lo memorizza. Per ogni file aperto vengono scritti 5 caratteri nel buffer a partire dalla posizione 1 e viene stampato sulla stampante (aperta come file logico 7) il numero del buffer. Si vede che se si parte dopo un reset dell'unità il primo buffer assegnato è il numero 3, mentre, se si parte dopo LOAD senza spegnere l'unità disco, il primo

buffer assegnato risulta il numero 4. Dalla linea 500 alla 550 vengono letti i primi 8 byte dei blocchi di 256 byte di memoria a partire dall'indirizzo 0 fino all'indirizzo 2048. Si scopre che i blocchi di memoria usati per i buffer sono i seguenti:

Buffer 4 indirizzo 1792 (0700H)
Buffer 3 indirizzo 1536 (0600H)
Buffer 2 indirizzo 1280 (0500H)
Buffer 1 indirizzo 1024 (0400H)
Buffer 0 indirizzo 0768 (0300H)

L'assegnazione della memoria ai canali varia a seconda della disponibilità, mentre ogni buffer corrisponde sempre alla stessa zona di memoria.

CAPITOLO 7

SISTEMA OPERATIVO E INTERPRETE BASIC

7.1 SISTEMA OPERATIVO

Gli ultimi 8K di memoria ROM, a partire dall'indirizzo 57344 (E000H), contengono il sistema operativo del VIC; gli 8K precedenti, da 49152 (C000H), contengono l'interprete BASIC. Il sistema usa le locazioni in pagina zero, da 144 a 255 (0090H-00FFH) per le sue variabili; quelle in pagina 1, da 256 a 511 (0100H-01FFH), per l'area di stack; quelle in pagina 2, da 601 a 767 (0259H-02FFH), per altre variabili che non devono necessariamente stare in pagina zero; quelle in pagina 3, da 788 a 8219 (0314H-0333H), per i vettori dei salti indiretti. Da 828 a 1019 (033CH-03FBH) è situato il buffer per la gestione del nastro.

Per i programmatori esperti è utile conoscere il significato delle variabili usate dal sistema operativo e gli indirizzi delle routine del sistema. Infatti, grazie alla tabella dei vettori dei salti indiretti situata in RAM, essi possono intervenire sul sistema e modificare alcune routine per realizzare funzioni particolari. Un esempio di questo si ha nel paragrafo 4 del capitolo 8, dove viene fatta un'aggiunta alla routine di interrupt. Procedure di questo tipo si possono rivelare necessarie per gestire apparecchiature speciali di input/output collegate alla porta utente.
Le routine del sistema operativo del VIC prendono il nome di ROUTINE KERNAL e la tabella dei salti indiretti si chiama KERNAL JUMP TABLE. La tecnica dei salti indiretti tramite tabella dà una grande flessibilità: essa consente agli utenti di intervenire sul sistema e li garantisce verso eventuali modifiche che la casa produttrice può apportare al sistema operativo stesso; infatti il riferimento viene fatto attraverso la tabella dei salti e le cose continuano a funzionare anche se è stato cambiato qualche indirizzo in una nuova ROM.

TAVOLA VARIABILI DEL SISTEMA		
DEC.	ESAD.	SIGNIFICATO
144	0090	Parola di stato ST
145	0091	Flag PIA:STOP e RVS

DEC.	ESAD.	SIGNIFICATO
146	0092	Cost. temp. nastro
147	0093	Flag: Load=0 Verify=1
148	0094	Flag car. output seriale
149	0095	Buffer car. output seriale
150	0096	Flag EOT da nastro
151	0097	Registro salvataggio
152	0098	Numero file aperti
153	0099	Device input (norm.0)
154	009A	Device output (norm.3)
155	009B	Parità car. nastro
156	009C	Flag byte ricevuto
157	009D	Flag output 128 se diretto, 0 se RUN
158	009E	Flag err. passo 1 nastro
159	009F	Flag err. passo 2 nastro
160	00A0	3 byte CLOCK
163	00A3	FLAG EOI cont. bit seriali
164	00A4	Cont. cicli I/O seriale
165	00A5	Cont. bit scritti/testata
166	00A6	Puntatore buffer nastro
167	00A7	Cont. scritt. nastro/lettura
168	00A8	Scritt. byte/lettura err.
169	00A9	Flag scritt./lett. nastro
170	00AA	Buffer byte ricev.
171	00AB	Contr. parità byte ricev.
172	00AC	2 byte: punt. buffer nastro/inizio progr.
174	00AE	2 byte: fine progr./punt. fine nastro
176	00B0	2 byte: contatore oper. nastro
178	00B2	2 byte: punt. buffer nastro
180	00B4	Timer nastro (1=abilitato)
181	00B5	EOT nastro/invio bit seguente
182	00B6	Err. lettura/buffer output
183	00B7	Lungh. stringa nome file
184	00B8	LFN file corrente
185	00B9	SA operazione corrente
186	00BA	Numero device corrente
187	00BB	2 byte: punt. nome file
189	00BD	Parola scritt./car. input
190	00BE	Num. blocchi da legg./scriv.
191	00BF	Buffer seriale
192	00C0	Switch motore cassetta
193	00C1	2 byte: ind. part. I/O
195	00C3	2 byte: punt. KERNAL
197	00C5	Tasto premuto

DEC.	ESAD.	SIGNIFICATO
198	00C6	Num. car. buffer tastiera
199	00C7	Flag RVS: 0=off
200	00C8	Punt. fine linea input
201	00C9	2 byte: pos. cursore, riga-colonna
203	00CB	Num. tasto: 64= nessun tasto
204	00CC	Flag lamp. curs.: 0=on 1=off
205	00CD	Tempo lamp. cursore
206	00CE	Car. sotto il cursore
207	00CF	Flag. fase lamp. curs.
208	00D0	Flag input: video/tastiera
209	00D1	2 byte: punt. linea video
211	00D3	Pos. curs. linea corrente
212	00D4	Flag curs.:0=dir. 1=progr.
213	00D5	Lungh. linea corrente
214	00D6	Numero linea corrente
215	00D7	Buffer tasto premuto
216	00D8	Flag inserimenti
217	00D9	24 byte: tabella linee video
241	00F1	Ultimo byte linee video
242	00F2	Punt. temporaneo linea video
243	00F3	2 byte: punt. colore video
245	00F5	2 byte: punt. tabella car. tastiera
247	00F7	2 byte: punt. buffer ricez. RS232
249	00F9	2 byte: punt. buffer trasm. RS232
251	00FB	4 byte: spazio libero
255	00FF	Memoria Basic
256	0100	256 byte: area stack del sistema
601	0259	10 byte: tabella lfn (file logici)
611	0263	10 byte: tabella numeri device
621	026D	10 byte: tabella indirizzi secondari
631	0277	10 byte: buffer tastiera
641	0281	2 byte: inizio memoria (START)
643	0283	2 byte: fine memoria (TOP)
645	0285	Flag timeout IEEE seriale
646	0286	Codice colore corrente
647	0287	Colore sotto il cursore
648	0288	Num. pagina mappa video
649	0289	Mass. dim. buffer tastiera
650	028A	Flag ripet. tasti (0=curs. 255=tutti)
651	028B	Cont. veloc. ripet. tasti
652	028C	Interv. ripet. tasti
653	028D	Flag tastiera
654	028E	Immag. ultimo tasto spostato

DEC.	ESAD.	SIGNIFICATO
655	028F	2 byte: Punt. logica decodifica tastiera
657	0291	Switch SHIFT: 0=disab. 128=abilit.
658	0292	Flag scroll: 0=on
659	0293	Reg. contr. RS 232
660	0294	Reg. com. RS 232
661	0295	2 byte: non standard tempo bit/2-100
663	0297	Reg. stato RS 232
664	0298	Num. bit da mandare
665	0299	2-byte: num. bit/sec
667	029B	Punt. ricez. RS 232
668	029C	Punt. input RS 232
669	029D	Punt. trasm. RS 232
670	029E	Punt. output RS 232
671	029F	2 byte: Mem. IRQ durante op. nastro
673	02A1	95 byte: contengono 0 all'inizio
780	030C	Mem. temp. reg. A 6502
781	030D	Mem. temp. reg. X 6502
782	030E	Mem. temp. reg. Y 6502
783	030F	Mem. temp. reg. P 6502
784	0310	4 byte
788	0314	2 byte: IRQ (60095 (EABFH))
790	0316	2 byte: BRK (65234 (FED2H))
792	0318	2 byte: NMI (65197 (FEADH))
794	031A	2 byte: OPEN (62474 (F40AH))
796	031C	2 byte: CLOSE (62282 (F34A))
798	031E	2 byte: Set dev. input (62151 (F2C7H))
800	0320	2 byte: Set dev. output (62217 (F309H))
802	0322	2 byte: Reset vett. I/O (62451 (F3F3H))
804	0324	2 byte: INPUT (61966 (F20EH))
806	0326	2 byte: OUTPUT (62074 (F27AH))
808	0328	2 byte: STOP (63344 (F770H))
810	032A	2 byte: GET (61941 (F1F5H))
812	032C	2 byte: Abort I/O (62447 (F3EFH))
814	032E	2 byte: USR (65234 (FED2H))
816	0330	2 byte: LOAD (62793 (F549H))
818	0332	2 byte: SAVE (63109 (E685H))

Le 16 routine che hanno l'indirizzo memorizzato nei byte della RAM da 780 a 819, di cui è stato elencato tra parentesi l'indirizzo decimale ed esadecimale, possono essere "intercettate" dal programmatore per fare eseguire funzioni diverse dalle solite (oltre le solite); le routine scritte dall'utente in linguaggio macchina devono avere il loro indirizzo di inizio memorizzato nella coppia di byte relativa e terminare con un salto all'indirizzo che vi era precedentemente contenuto. L'utente può,

usando per esempio VICMON, disassemblare le routine che gli interessano e studiare cosa fanno. Non si tratta di lavori difficili, ma richiedono un pò di esperienza e molta pazienza.

L'utente può anche usare direttamente altre routine del sistema operativo, pur di conoscerne l'indirizzo di inizio e il modo di operare. Anche per queste routine può essere usato VICMON per disassemblarle e studiarle. Si elencano alcune routine del sistema operativo, riportando l'indirizzo di inizio, i registri coinvolti per i dati di ingresso e di uscita e quello che è necessario sapere per poterle adoperare. Per ognuna di queste routine viene usato un nome mnemonico al solo scopo di distinguerle una dall'altra in modo semplice; nei programmi in linguaggio macchina si accede alle routine con una istruzione JSR avente come campo operando l'indirizzo della routine. Per una buona gestione è necessario controllare gli errori che possono determinarsi usando le routine stesse. Quando , in ogni routine, viene incontrata l'istruzione RTS, il controllo ritorna al programma dell'utente (all'istruzione seguente la JSR di chiamata).

TABELLA ROUTINE KERNAL

Nome	Ind.tab.JMP		Oper.JMP		Scopo
	dec.	esad.	dec.	esad.	
ACPTR	65445	FFA5	61209	EF19	input porta seriale
CHKIN	65478	FFC6	(798)	(031E)	OPEN canale input
CHKOUT	65481	FFC9	(800)	(0320)	OPEN canale output
CHRIN	65487	FFCF	(804)	(0324)	input canale
CHROUT	65490	FFD2	(806)	(0326)	output canale
CIOUT	65448	FFA8	61156	EEE4	output porta seriale
CLALL	65511	FFE7	(812)	(032C)	CLOSE canali e file
CLOSE	65475	FFC3	(796)	(031C)	CLOSE file logico
CLRCHN	65484	FFCC	(802)	(0322)	CLOSE canali I/O
GETIN	65512	FFE4	(810)	(032A)	GET car. buff. tast.
IOBASE	65523	FFF3	58624	E500	ind. base dev. I/O
LISTEN	65457	FFB1	60951	EE17	LISTEN dev. bus ser.
LOAD	65493	FFD5	62786	F542	LOAD
MEMBOT	65436	FF9C	65154	FE82	Read/Set inizio mem.
MEMTOP	65433	FF99	65139	FE73	Read/Set fine mem.
OPEN	65472	FFC0	(794)	(031A)	OPEN lfn
PLOT	65520	FFF0	58634	E50A	Read/Set pos. curs.
RDTIM	65502	FFDE	63328	F760	Legge il clock
READST	65463	FFB7	65111	FE57	Legge par. stato I/O
RESTOR	65415	FF8A	64850	FD52	Ripristino vett. e I/O
SAVE	65496	FFD8	63093	F675	SAVE

Nome	Ind.tab.JMP		Oper.JMP		Scopo
	dec.	esad.	dec.	esad.	
SCNKEY	65439	FF9F	60190	EB1E	Scansione tastiera
SCREEN	65517	FFED	58629	E505	X,Y org. video
SECOND	65427	FF93	61120	EEC0	SA dopo LISTEN
SETLFS	65466	FFBA	65104	FE50	Set ind. logici
SETMSG	65424	FF90	65126	FE66	Messaggi KERNAL
SETNAM	65469	FFBD	65097	FE49	Set nome file
SETTIM	65499	FFDB	65097	FE49	Set clock
SETTMO	65442	FFA2	65135	FE6F	Set timeout bus ser.
STOP	65505	FFE1	(808)	(0328)	STOP
TALK	65460	FFB4	60948	EE14	TALK bus ser.
TKSA	65430	FF96	61134	EECE	SA dopo TALK
UDTIM	65514	FFEA	63284	F734	Incr. clock
UNLSN	65454	FFAE	61188	EF04	UNLISTEN bus ser.
UNTLK	65451	FFAB	61174	EEF6	UNTALK bus ser.
VECTOR	65412	FF8D	64855	FD57	Read/Set vettore I/O

La tabella precedente reca il nome mnemonico delle routine, l'indirizzo decimale e esadecimale della tabella dei salti (sono occupati 3 byte con il codice operativo JMP e un indirizzo, che risulta indiretto quando compare tra parentesi), l'indirizzo decimale e esadecimale del salto e una breve spiegazione dello scopo. Gli indirizzi dei salti sono stati ottenuti con la funzione D di VICMON. Come si può vedere, alcune di queste routine rimandano agli indirizzi già visti in RAM, cioè alcune routine sono intercettabili.

Segue ora la descrizione delle caratteristiche delle routine (gli indirizzi di chiamata sono riportati in esadecimale).

ACPTR: legge un byte dal bus seriale.
Chiamata: FFA5 Comunic.: reg. A
Preparazione: TALK, TKSA Errore: READST
Byte stack: 13 Reg. usati: A e X
Si deve usare anche TKSA se è necessario un indirizzo secondario.

CHKIN: apre un canale per input.
Chiamata: FFC6 Comunic.: reg. X
Preparazione: OPEN
Errore: 3=file non aperto
5=device non presente
6=file non di input
Reg. usati: A e X
Definisce un file logico già aperto come un canale di input. Si deve usare prima di

CHRIN o di GETIN. Per ricevere dalla tastiera, se non è aperto alcun altro canale di input, si può evitare di chiamare sia la OPEN che questa routine. Se viene usata con il bus seriale, essa invia automaticamente l'indirizzo di TALK, e anche SA, se specificato nella OPEN. Il numero LFN deve essere scritto nel registro X prima della chiamata.

CHKOUT: apre un canale per output.
Chiamata: FFC9 Comunic.: reg. X
Preparazione: OPEN
Errore: come la precedente Reg. usati: A e X
Definisce un file logico già aperto come un canale di output. Non si deve usare se l'output è diretto al video e non sono aperti altri canali di output (come anche OPEN). Se usata con il bus seriale, essa invia automaticamente l'indirizzo di LISTEN specificato nella OPEN, e anche SA, se è presente, LFN deve essere precedentemente scritto in X.

CHRIN: legge un carattere dal canale di input.
Chiamata: FFCF Comunic.: reg. A
Preparazione: OPEN, CHKIN Errore: READST
Reg. usati: A e X
Deve essere usata dopo OPEN e CHKIN, a meno che non legga da tastiera. Il byte si trova in A. Il canale rimane aperto. Il comportamento per l'input da tastiera è il seguente: la linea scritta va nel buffer quando si preme il RETURN. Si chiama la routine per prelevare dal buffer un carattere per volta controllando se si arriva al RETURN.

CHROUT: scrive un carattere.
Chiamata: FFD2 Comunic.: reg. A
Preparazione: OPEN, CHKOUT Errore: READST
Manda un carattere su un canale già aperto. Si deve usare prima OPEN e CHKOUT; se non è così il carattere viene inviato al dispositivo 3, il video. Il carattere va posto in A. Il canale resta aperto. Se sul bus seriale sono aperti più dispositivi, il dato viene inviato a tutti; per questo bisogna lasciare aperto solo quello che interessa e chiudere gli altri.

CIOUT: invia un byte sul bus seriale.
Chiamata: FFA8 Comunic.: reg. A
Preparazione: LISTEN, SECOND Errore: READST
Il dispositivo deve essere in stato LISTEN altrimenti si ha errore di timeout. Se necessario SA, deve essere usata prima la SECOND. Il carattere da inviare deve essere posto in A. La routine mantiene un carattere nel buffer. Quando viene chiamata UNLSN viene inviato il carattere che sta nel buffer insieme a EOI e il dispositivo esce dallo stato di attesa.

CLALL: chiude tutti i file.
Chiamata: FFE7 Byte stack: 11
Reg. usati: A e X
Viene resettato il puntatore alla tabella dei file aperti chiudendo tutti i file e resettando tutti i canali di I/O.

CLOSE: chiude un file logico.
Chiamata: FFC3 Comunic.: A
Reg. usati: A e X
Si deve caricare in A il LFN del file da chiudere e chiamare la routine.

CLRCHN: chiude tutti i canali e rimette i canali di I/O ai valori normali (0 per la tastiera e 3 per il video).
Chiamata: FFCC Byte stack: 9
Reg. usati: A e X
Se un canale da chiudere riguarda la porta seriale, viene mandato o un UNTALK per input o un UNLISTEN per output. Si ricordi che se sulla porta seriale sono attivi più dispositivi, essi ricevono contemporaneamente dati.

GETIN: preleva un carattere dal buffer tastiera.
Chiamata: FFE4 Comunic.: A
Reg. usati: A e X
Il carattere prelevato dal buffer si ritrova in A in codice ASCII. Se non ci sono caratteri A contiene zero. I caratteri sono messi nel buffer dalla routine SCNKEY che va in onda per effetto dell'interrupt 60 volte al sec.; il buffer può contenere 10 caratteri, quelli eccedenti vanno persi.

IOBASE: fornisce l'indirizzo della pagina in cui sono mappati gli indirizzi di I/O.
Chiamata: FFF3 Comunic.: X e Y
Byte stack: 2 Reg. usati: X e Y
Dopo la chiamata X contiene il byte basso e Y il byte alto dell'indirizzo. Usando questo indirizzo più un offset (spostamento) si può accedere alle locazioni che interessano per una periferica.

LISTEN: predispone una periferica a ricevere dati dal bus seriale.
Chiamata: FFB1 Comunic.: A
Errore: READST Reg. usati: A
Prima della chiamata si deve porre in A un numero tra 4 e 30; esso viene decodificato per ottenere l'indirizzo della periferica; questo le viene inviato e la pone in stato di attesa.

LOAD: trasferisce byte da una periferica alla memoria del VIC.
Chiamata: FFD5 Comunic.: A, X, Y
Preparazione: SETLFS, SETNAM Reg. usati: A, X e Y

Errori: 4=file non trovato
5=device non presente
8=manca nome file
9=numero device non legale

Prima della chiamata si devono usare SETLFS e SETNAM e caricare A con 0 per LOAD o con 1 per VERIFY. Se il file è stato aperto con SA = 0 (rilocazione) vengono ignorate le informazioni di testata del file, ma i registri X e Y devono contenere l'indirizzo di inizio memorizzazione. Se invece SA è 1, 2 o 3 l'indirizzo di inizio memorizzazione viene prelevato dalla testata del file. Dopo l'esecuzione X e Y contengono l'indirizzo dell'ultima locazione RAM caricata.

MEMBOT: definisce l'inizio della memoria RAM.
Chiamata: FF9C Comunic.: X e Y
Reg. usati: X e Y Byte stack: 2
Può essere usata in 2 modi: ponendo il bit di Carry a 1 si ha in X e Y l'indirizzo di inizio RAM, cioè una lettura; ponendo il bit di Carry a 0 si scrive nei puntatori all'inizio RAM il valore contenuto in X e Y (X byte basso e Y byte alto).

MEMTOP: definisce la fine della memoria.
Chiamata: FF99 Comunic.: X e Y
Reg. usati: X e Y Byte stack: 2
Può essere usata in 2 modi: con Carry a 1 si legge in X e Y l'indirizzo di fine memoria; con Carry a 0 si scrive nei puntatori alla fine della memoria il contenuto dei registri X e Y. In questa seconda forma si usa per lasciare della memoria libera agli indirizzi alti per programmi in linguaggio macchina.

OPEN: apre un file logico.
Chiamata: FFC0 Reg. usati: A, X e Y
Reparazione: SETLFS, SETNAM
Errore: 1=troppi file aperti
2=file già aperto
4=file non trovato
5=device non presente
8=manca nome file

La routine non ha parametri e lavora su quelli sistemati dalle due routine che devono essere chiamate prima: SETNAM e SETLFS. Dopo questa routine si può lavorare su un file logico che risulta completamente definito.

PLOT: riguarda la posizione del cursore sul video.
Chiamata: FFF0 Comunic.: A, X e Y
Byte stack: 2 Reg. usati: A, X e Y
Può essere usata in due modi. Il reg. X si riferisce alla riga (0-21), Y alla colonna (0-22). Se Carry=1 legge in X e Y la posizione corrente del cursore, se Carry=0 scrive, cioè sposta il cursore nella posizione determinata da X e Y.

RDTIM: legge il contenuto dell'orologio.
Chiamata: FFDE Comunic.: A, X e Y
Byte stack: 2 Reg. usati: A, X e Y
L'orologio è un contatore che occupa 3 byte e viene aggiornato 60 volte al secondo. All'uscita si ha: byte più significativo in A, successivo in X e meno significativo in Y.

READST: legge la parola di stato.
Chiamata: FFB7 Comunic.: A
Byte stack: 2 Reg. usati: A
Trasferisce in A la parola di stato, chiamandola dopo un operazione di I/O si sa se ci sono stati errori.
I bit di ST hanno il significato seguente:

Pos.bit	Valore	Cassette lettura	I/O ser.	Cassette Ver./Load
0	1		timeout scritt.	
1	2		timeout lett.	
2	4	blocco corto		blocco corto
3	8	blocco lungo		blocco lungo
4	16	fatale		incompat.
5	32	checksum		checksum
6	64	fine file	EOI	
7	128	fine nastro	device no pres.	fine nastro

RESTOR: ripristina i valori iniziali in tutti i vettori di interrupt usati dal sistema.
Chiamata: FF8A Byte stack: 2
Reg. usati: A, X e Y
Serve per ripristinare le condizioni iniziali del sistema.

SAVE: salva una parte di memoria su una periferica.
Chiamata: FFD8 Comunic.: A, X e Y
Preparazione: SETLFS, SETNAM Reg. usati: A, X e Y
Errore: 5=device non presente
8=manca nome file
9=numero device non legale
Copia un pezzo di memoria il cui indirizzo di inizio in pagina zero è puntato dal contenuto di A, mentre quello di fine è puntato dal contenuto di X e Y. Prima si devono usare SETNAM e SETLFS per definire il file di memorizzazione. Il nome del file può essere omessso se il salvataggio viene effettuato su cassetta. Non può essere usato come numero di dispositivo 0 (tastiera) e 3 (video). Prima di chiamare la routine si deve scrivere, in due locazioni contigue della pagina zero, l'indirizzo di inizio della memoria da salvare e porre in A il puntatore alla prima di queste due locazioni. Si ricordi che gli indirizzi si scrivono anteponendo il byte basso a quello alto.

SCNKEY: scandisce la tastiera.
Chiamata: FF9F Reg. usati: A, X e Y
Scandisce la tastiera e pone nel buffer il codice ASCII del carattere che corrisponde al tasto premuto. Il buffer della tastiera va da 631 a 640 (0277H-0280H). Viene chiamata 60 volte al secondo dalla routine di interrupt.

SCREEN: fornisce il formato del video.
Chiamata: FFED Comunic.: X e Y
Byte stack: 2 Reg. usati: X e Y
Il numero delle righe del video (22) va in X e il numero di colonne (23) va in Y. Può essere usata per adattare programmi alla configurazione del video del VIC.

SECOND: invia l'indirizzo secondario SA.
Chiamata: FF93 Comunic.: A
Preparazione: LISTEN Errore: READST
Reg. usati: A
Deve essere usata dopo LISTEN se per il dispositivo è necessario un indirizzo secondario. Se il dispositivo è sul bus seriale, SA deve essere modificato con OR 96 (60H) prima di usarlo; deve essere posto in A.

SETLFS: prepara i parametri per un file logico.
Chiamata: FFBA Communic.:A, X e Y
Byte stack: 2
Prepara il LFN, l'indirizzo della periferica e SA per le altre routine KERNAL. LFN

è usato come chiave di ricerca nella tabella dei file creata dalla OPEN. Gli indirizzi delle periferiche vanno da 0 a 30 e gli indirizzi superiori a 4 si riferiscono automaticamente al bus seriale. Gli indirizzi usati sono i seguenti: 0 per tastiera, 1 per cassetta, 2 per RS-232C, 3 per video, 4 per bus seriale (stampante), 8 per bus seriale (disco). Prima della chiamata si devono caricare i registri così: LFN in A, numero device in X, SA in Y (se SA non serve Y deve contenere 255).

SETMSG: controlla la stampa dei messaggi di controllo e di errore.
Chiamata: FF90 Comunic.: A
Byte stack: 2 Reg. usati: A
Il messaggio stampato dipende dal numero che si trova in A. Se il bit 6 è 1 i messaggi sono di controllo, se il bit 7 è 1 i messaggi sono di errore.

SETNAM: prepara il nome del file.
Chiamata: FFBD Comunic.:A, X e Y
Si deve porre in A la lunghezza del nome del file; in X e Y l'indirizzo del nome del file (byte basso prima del byte alto). Se non è richiesto il nome, si deve mettere 0 in A.

SETTIM: serve per inizializzare l'orologio interno.
Chiamata: FFDB Comunic.: A,X,Y
Byte stack: 2
L'orologio del sistema viene gestito dalla routine di interrupt che va in onda 60 volte al secondo. E' costituito da 3 byte consecutivi e questo gli permette di contare fino a 5.184.000 sessantesimi di secondo, cioè 24 ore; a quel punto riparte da 0. Con questa routine si può caricare il numero desiderato nell'orologio, ponendolo nei 3 registri A, X e Y, in ordine dal byte più significativo al meno significativo.

SETTMO: attiva il flag per il controllo del tempo di risposta sul bus seriale.
Chiamata: FFA2 Comunic.: A
Byte stack: 2
Se il flag di timeout è attivato il VIC attende una risposta sul bus seriale per 64 millisecondi; dopo questo tempo il colloquio è interrotto e si ha segnalazione di errore. Al momento della chiamata se il bit 7 di A è 0 viene abilitato il flag, se è 1 esso viene disabilitato. Viene usato il timeout per segnalare che non è stato trovato un file da una OPEN.

STOP: controlla se è premuto il tasto STOP.
Chiamata: FFE1 Comunic.:A
Reg. usati: A, X

Dopo la chiamata, se è stato trovato premuto lo STOP, si ha il flag Z in on. Se non è stato premuto i flag non vengono modificati e in A si trova un byte che rappresenta l'ultima riga della tastiera che è stata analizzata.

TALK: invia un comando di TALK attraverso il bus seriale.
Chiamata: FFB4 Comunic.: A
Errore: READST Reg. usati: A
Prima della chiamata si deve porre in A il numero del dispositivo (da 4 a 30). Questo numero viene convertito nell'indirizzo del dispositivo e inviato al bus seriale.

TKSA: manda SA a un dispositivo che ha ricevuto il comando TALK.
Chiamata: FF96 Comunic.: A
Preparazione: TALK Errore :READST
Reg. usati: A
Prima della chiamata SA deve essere posto in A. Può essere usata solo dopo TALK.

UDTIM: aggiorna l'orologio interno.
Chiamata: FFEA Byte stack: 2
Reg. usati: A, X
E' chiamata dalla routine di interrupt di sistema 60 volte al secondo. Se l'utente gestisce le interruzioni in modo diverso, deve chiamarla ugualmente per aggiornare l'orologio, come anche deve chiamare la routine di STOP per far sì che il tasto corrispondente resti funzionante.

UNLSN: invia un comando di UNLISTEN a tutti i dispositivi del bus seriale.
Chiamata: FFAE Errore: READST
Reg. usati: A
Questo comando riguarda solo i dispositivi che si trovano in stato di LISTEN.

UNTLK: invia un comando di UNTALK a tutti i dispositivi del bus seriale.
Chiamata: FFAB Errore: READST
Reg. usati: A
Questo comando riguarda solo i dispositivi che sono in stato TALK.

VECTOR: aggiorna tutti i vettori del sistema che stanno in RAM.
Chiamata: FF8D Comunic.: X, Y

Byte stack: 2 Reg. usati: A, X e Y
Il funzionamento della routine dipende dal bit Carry. Se Carry=1 la routine usa l'indirizzo contenuto in X e Y come puntatore a una lista in RAM dove memorizza tutti i vettori del sistema. Se invece Carry=0 la lista puntata da X e Y serve per andare a scrivere nei vettori in RAM modificandone il contenuto. Conviene chiamare una prima volta la routine per leggere in area utente i contenuti dei registri, poi modificare quelli che servono e chiamare una seconda volta la routine per andare a riscrivere i vettori.

Si riporta il listato della routine di POWER UP, ottenuta con la funzione D di VICMON lanciato dopo aver trasferito l'uscita sulla stampante con :OPEN4,4:CMD4. La routine inizia in 64802 (FD22H).

```
B*
   PC  SR AC XR YR SP
.;603E 33 00 63 00 F6
.

., FD22 LDX #$FF
., FD24 SEI
., FD25 TXS
., FD26 CLD
., FD27 JSR $FD3F
., FD2A BNE $FD2F
., FD2C JMP ($A000)
., FD2F JSR $FD8D
., FD32 JSR $FD52
., FD35 JSR $FDF9
., FD38 JSR $E518
., FD3B CLI
., FD3C JMP ($C000)
.
```

Commenti:
.da FD22 a FD25 viene inizializzato lo stack pointer e settato il flag dell'interrupt (disabilita l'interrupt)
.a FD26 viene azzerato il flag decimale
.a FD27 si ha un salto alla routine che controlla se è presente un cartridge di ROM all'indirizzo A000
.a FD2A prosegue l'inizializzazione da FD2F, se non è presente la ROM
.a FD2C salta all'indirizzo contenuto in A000, se presente una ROM, dopo aver verificato se i 5 byte da A004H ad A008H contengono la stringa "A0CBM". I byte A000H e A001H contengono l'indirizzo di inizio del programma contenuto nella ROM aggiunta e i byte A0002H e A0003H contengono l'indirizzo per tornare il controllo al BASIC quando si preme RUN/STOP e RESTORE
.a FD2F salta alla routine che controlla se è presente una espansione RAM
.a FD32 salta alla routine che inizializza i vettori di salto (0314H-0333H)
.a FD35 salta alla routine che inizializza gli integrati 6522

.a FD38 salta alla routine che pone valori di default nelle variabili del sistema
.a FD3B resetta i flag di interrupt (abilita gli interrupt)
.a FD3C salta al BASIC.

7.2 INTERPRETE BASIC

Anche l'interprete BASIC usa delle locazioni in pagina zero per le variabili e le aree di lavoro necessarie; segue la relativa tabella.

DEC.	ESAD.	SIGNIFICATO
0	0000	3 byte: usati dalla funzione USR
3	0003	2 byte: conversione floating-point/interi
5	0005	2 byte: conversione interi/floating-point
7	0007	contatore Basic
8	0008	scansione stringa tra virgolette
9	0009	posizione cursore nella linea
10	000A	0=LOAD, 1=VERIFY
11	000B	puntatore buffer input/num. indici
12	000C	flag per DIM
13	000D	flag variab. FFH=stringa, 00=numerica
14	000E	tipo num. 80H=interi, 00=floating
15	000F	flag: DATA/LIST/memoria
16	0010	flag:indici/funzioni
1	0011	0=INPUT, 40H=GET, 98H=READ
18	0012	flag:segno ATN/confronti
19	0013	dispositivo attuale di I/O
20	0014	2 byte: indirizzi interi Basic
22	0016	punt. stringa
23	0017	2 byte: vettore temp. stringa
25	0019	9 byte: stack vett. temp. stringa
34	0022	4 byte: area puntatori
38	0026	5 byte: area per moltiplicazione
43	002B	2 byte: puntatore inizio Basic
45	002D	2 byte: puntatore inizio variabili
47	002F	2 byte: puntatore inizio array
49	0031	2 byte: puntatore fine array
51	0033	2 byte: puntatore area stringhe (da indirizzo alto)
53	0035	2 byte: puntatore lavoro stringhe
55	0037	2 byte: puntatore fine memoria
57	0039	2 byte: num. linea corrente Basic
59	003B	2 byte: num. linea precedente

DEC.	ESAD.	SIGNIFICATO
61	003D	2 byte: punt. Basic per CONT
63	003F	2 byte: linea corrente DATA
65	0041	2 byte: indirizzo corrente DATA
67	0043	2 byte: vettore di input
69	0045	2 byte: nome variab. corrente
71	0047	2 byte: ind. variabile corrente
73	0049	2 byte: punt. var. per FOR/NEXT
75	004B	2 byte: salvat. Y/operaz. /punt.
77	004D	comparazione simboli
78	004E	6 byte: area lavoro
84	0054	3 byte: vettore salto per funzioni
87	0057	10 byte: area lavoro numeri
97	0061	acc.1: esponente
98	0062	4 byte: acc.1: mantissa
102	0066	acc.1: segno
103	0067	puntatore calcolo serie
104	0068	acc.1: overflow
105	0069	6 byte: acc.2
111	006F	confr. acc. 1/acc. 2
112	0070	acc.1 arrotondamento
113	0071	2 byte: punt. buffer cassetta
115	0073	24 byte: routine GET
		122/123 (7AH-7BH) puntatori interni al buffer tastiera
139	008B	5 byte: area lavoro e memorizzaz. RND

Si fa notare che il sistema usa 6 byte per ACC.1, da 97 a 102 (0061H - 0066H), e 6 byte per ACC.2, da 105 a 110 (0069H - 006EH), e inoltre il byte 111 (006FH) per il confronto del segno tra i due accumulatori, e il byte 112 (0070H) per l'arrotondamento di ACC.1. Il BASIC esegue in generale i calcoli in floating point, ed usa questi due accumulatori per i calcoli in floating-point.

In pagina 2 vengono usate le locazioni da 512 a 600 (0200H-0258H) come buffer di input del Basic di 89 caratteri; un puntatore alla posizione nel buffer si trova nella routine che sta in pagina zero e parte da 115 (0073H), nei byte 122 e 123 (007AH-007BH).

La tabella dei "tokens" si trova da 49310 a 49567 (C09EH-C19FH); il codice del token -127 fa da puntatore alla tabella delle parole corrispondenti. Segue il programma TOKENS1, con il quale si stampano le parole chiave, corrispondenti ai tokens, memorizzate nella tabella. Alla linea 15 viene letto in L il valore del byte; se

esso è maggiore di 127 si sottrae 128, si stampa il carattere corrispondente e si fa uno spazio. Se il valore letto non supera 127 si stampa il carattere corrispondente non seguito da spazio.

```
1 REM TOKENS1
5 OPEN4,4:CMD4
10 FORK=49310TO49567
15 L=PEEK(K)
20 IFL<=127THENPRINTCHR$(L);:GOTO30
25 L=L-128:PRINTCHR$(L);" ";
30 NEXTK:PRINT
40 PRINT#4:CLOSE4
```

Risultati del programma TOKENS1 su stampante:

```
END FOR NEXT DATA INPUT# INPUT DIM READ LET GOTO RUN IF RESTORE GOSUB RETURN REM
 STOP ON WAIT LOAD SAVE VERIFY DEF POKE PRINT# PRINT CONT LIST CLR CMD SYS OPEN
CLOSE GET NEW TAB( TO FN SPC( THEN NOT STEP + - * / ↑ AND OR > = < SGN INT ABS U
SR FRE POS SQR RND LOG EXP COS SIN TAN ATN PEEK LEN STR$ VAL ASC CHR$ LEFT$ RIGH
T$ MID$ GO TO
```

Segue il programma TOKENS2 che, invece, stampa il contenuto dei byte della tabella come sono cioè i valori numerici letti.

```
1 REM TOKENS2
5 OPEN4,4:CMD4
10 FORK=49310TO49567
20 PRINTPEEK(K);
30 NEXTK:PRINT
40 PRINT#4:CLOSE4
```

Risultati programma TOKENS2 su stampante:

```
 69  78  196  70  79  210  78  69  88  212  68  65  84  193  73  78  80  85  84
 163  73  78  80  85  212  68  73  205  82  69  65  196  76  69  212  71  79  84
  207  82  85  206  73  198  82  69  83  84  79  82  197  71  79  83  85  194  8
2  69  84  85  82  206  82  69  205  83  84  79  208  79  206  87  65  73  212
76  79  65  196  83  65  86  197  86  69  82  73  70  217  68  69  198  80  79
75  197  80  82  73  78  84  163  80  82  73  78  212  67  79  78  212  76  73
83  212  67  76  210  67  77  196  83  89  211  79  80  69  206  67  76  79  83
 197  71  69  212  78  69  215  84  65  66  168  84  207  70  206  83  80  67  1
68  84  72  69  206  78  79  212  83  84  69  208  171  173  170  175  222  65
78  196  79  210  190  189  188  83  71  206  73  78  212  65  66  211  85  83
210  70  82  197  80  79  211  83  81  210  82  78  196  76  79  199  69  88  20
8  67  79  211  83  73  206  84  65  206  65  84  206  80  69  69  203  76  69
206  83  84  82  164  86  65  204  65  83  195  67  72  82  164  76  69  70  84
 164  82  73  71  72  84  164  77  73  68  164  71  207  0  84  79
```

Per mezzo di alcuni indirizzi contenuti in pagina zero si può saltare a 6 routine del Basic; essi sono:

.768 (0300H) 2 byte: contengono l'indirizzo 50234 (C43AH) della routine che invia i messaggi di errore.

.770 (0302H) 2 byte: contengono l'indirizzo 50307 (C483H) della routine che gestisce una nuova linea del programma.

.772 (0304H) 2 byte: contengono l'indirizzo 50556 (C57CH) della routine che trasforma le parole chiave del linguaggio in token.

.774 (0306H) 2 byte: contengono l'indirizzo 50970 (C71AH) della routine che lista i token.

.776 (0308H) 2 byte: contengono l'indirizzo 51172 (C7E4H) della routine che esegue le linee Basic. Se si lista con VICMON la routine, si vede che la prima istruzione è un salto alla routine che si trova in pagina zero e inizia al byte 115 (0073H). Segue il listato di questa routine:

```
B*
   PC  SR AC XR YR SP
.;603E 33 00 63 00 F6
.

., 0073 INC $7A
., 0075 BNE $0079
., 0077 INC $7B
., 0079 LDA $0206
., 007C CMP #$3A
., 007E BCS $008A
., 0080 CMP #$20
., 0082 BEQ $0073
., 0084 SEC
., 0085 SBC #$30
., 0087 SEC
., 0088 SBC #$D0
., 008A RTS
.
```

Le istruzioni delle prime 3 righe incrementano i 2 byte interni (7A e 7B), che agiscono come puntatori al byffer, e consentono di caricare nell'accumulatore il contenuto di una posizione del buffer stesso. Alla linea in 7C il programma analizza se il carattere posto in accumulatore è ":" (fine istruzione) e in questo caso esce.

Alla linea 80 controlla se il carattere è uno spazio e in tale caso va ad analizzare il carattere seguente. Se il carattere non è nè ":" ne "spazio" prosegue da 0084. L'utente può modificare i 5 byte che vanno da 0084 a 0089 e inserire un salto a un suo particolare programma (in linguaggio macchina) posto in RAM. Questo programma può fare quello che si desidera e viene eseguito PRIMA di eseguire normalmente una linea BASIC; esso deve terminare in modo da ripristinare il contenuto precedente dei 5 byte modificati e ritornare alla routine in 0084, senza naturalmente aver modificati i puntatori che usa la routine originaria.

778 (030AH) 2 byte: contengono l'indirizzo 52870 (CE86H) della routine che esegue le operazioni aritmetiche.

Si elencano gli indirizzi di inizio di alcune routine del BASIC che l'utente può richiamare per far eseguire quello che fa con esse il Basic; si cita il nome della funzione, seguito dall'indirizzo decimale e da quello esadecimale senza il suffisso H. Si consiglia, se si desidera provarle, di ricavarne prima una lista con VICMON o in qualche altro modo.

ESECUZIONE FUNZIONI O COMANDI		
NEW	50754	C642
CLR	50784	C660
LIST	50844	C69C
FOR	51010	C742
RESTORE	51229	C81D
STOP END	51244	C82C
CONT	51287	C857
RUN	51313	C871
GOSUB	51331	C883
GOTO	51360	C8A0
RETURN	51410	C8D2
DATA	51435	C8EB
IF	51496	C928
REM	51515	C93B
ON	51531	C94B
LET	51621	C9A5
LET cont.	51756	CA2C
PRINT#	51840	CA80
CMD	51846	CA86
PRINT	51866	CA9A
PRINT stringhe	51998	CB1E

PRINT caratt.	52027	CB3B
GET	52091	CB7B
INPUT	52133	CBA5
INPUT cont.	52159	CBBF
READ (INPUT/GET)	52230	CC06
NEXT	52510	CD1E
OR	53222	CFE6
AND	53225	CFE9
DIM	53374	D07E
FRE	54141	D37D
DEF	54195	D3B3
FNx sint.	54241	D3E1
FNx valut.	54260	D3F4
STR$	54373	D465
CHR$	55020	D6EC
LEFT$	55040	D700
RIGHT$	55084	D72C
MID$	55095	D737
LEN	55164	D77C
VAL	55213	D7AD
PEEK	55309	D80D
POKE	55332	D824
WAIT	55341	D82D
LOG	55786	D9EA
SGN	56377	DC39
ABS	56408	DC58
INT	56524	DCCC
SQR	57201	DF71
EXP	57325	DFED
RND	57492	E094
COS	57953	E261
SIN	57960	E268
TAN	58033	E2B1
ATN	58123	E30B

INVIO MESSAGGI		
Messaggi Basic	49555	C193
Invia messaggio	50229	C435
READY	50292	C474
EXTRA IGNORED	52476	CCFC

REDO FROM START			
TYPE MISMATCH	52600		CD78
SYNTAX ERROR	53000		CF08
ILLEGAL DIRECT	54182		D3A6
OVERFLOW	55678		D97E
IN seguito da:	56770		DDC2
Numero linea	57781		DDCD
BYTES FREE....	58153		E429

ALTRE		
Indirizzo parole chiave	49152	C000
Indirizzo funzioni	49222	C046
Indirizzi operatori	49268	C074
Tabella parole chiave	49298	C092
Ricerca stack FOR/GOSUB	50058	C38A
Test su memoria disp.	50184	C408
Link linee Basic	50483	C533
Accetta linea da tast.	50528	C560
Ricerca un num. linea	50707	C613
Reset a partenza progr.	50830	C68E
Esecuz. frase Basic	51181	C7ED
Ricerca pross. istruz.	51462	C906
Ricerca pross. linea	51465	C909
Legge num. decimali	51563	C96B
Aggiunge cifra ACC 1	51741	CA1D
Dati di input non buoni	52045	CB4D
Car PROMPT per input	52217	CBF9
Ricerca e calcolo espress.	52638	CD9E
Calc. espr. in parentesi	52977	CEF1
Ricerca parent. destra	52983	CEF7
Ricerca parent. sinistra	52986	CEFA
Ricerca virgola	52989	CEFD
Prepara funz. da calcol.	53005	CF0D
Cerca nome var.	53012	CF14
Ident. funzione	53159	CFA7
Comparazione	53270	D016
Cerca locaz. var.	53387	D08B
Test car. alfab.	53523	D113
Creazione variabile	53533	D11D
Puntatore a var. con ind.	53652	D194
32768 in binario float.	53669	D1A5

Calc. espress. int. pos.	53674	D1AA
Cerca/crea var. indice	53713	D1D1
Calcola dim. indici	54092	D34C
Conv. in floating	54161	D391
Calcola vett. stringhe	54389	D475
Scandisce stringhe	54407	D487
Costruz. vett. stringhe	54516	D4F4
Sistem. memoria (garbage)	54566	D526
Sistem. stringa	54790	D606
Concatenaz. stringhe	54845	D63D
Costruz. stringhe	54906	D67A
Elimina stringhe	54947	D6A3
Pulisce stack	55003	D6DB
Toglie par. funz. da stack	55137	D761
Conv. stringhe in num.	55170	D782
Accetta param. a byte	55195	D79B
Accetta par. POKE/WAIT	55275	D7EB
Conv. float. in fixed	55287	D7F7
Arrot. in ACC 1 (+0.5)	55369	D849
Esegue sottrazione	55376	D850
Esegue addizione	55394	D862
Compl. ACC 1	55623	D947
Moltiplicazione a byte	55683	D983
Cost. per funzioni	55740	D9BC
Esegue moltiplic.	55856	DA30
Moltiplicazione a bit	55897	DA59
Carica ACC 2	55948	DA8C
Contr. ACC 1 e ACC 2	55991	DAB7
Tratta Overflow/underflow	56020	DAD4
Moltiplica per 10	56034	DAE2
10 in floating binario	56057	DAF9
Divide per 10	56062	DAFE
Esegue divisione into	56071	DB07
Esegue divisione by	56082	DB12
Carica ACC 1	56226	DBA2
Memorizza ACC 1	56263	DBC7
Copia ACC 2 in ACC 1	56316	DBFC
Copia ACC 1 in ACC 2	56332	DC0C
Arrot. ACC 1	56347	DC1B
Calcola segno ACC 1	56363	DC2B
Confronta ACC 1/Memoria	56411	DC5B
Conv. floating in fixed	56475	DC9B

Conv. stringa in float.	56563	DCF3
Preleva cifra ASCII	56702	DD7E
Costante 1E+9	56755	DDB3
Conv. TI$ in ASCII	56997	DDDD
Cost. conv. numeriche	57105	DF11
Eleva a potenza	57208	DF78
Negazione	57268	DFB4
Cost. valutaz. stringhe	57279	DFBF
Valutaz. funzioni	57408	E040
Tratt. cost. RND	57482	E08A
Modif. routine KERNAL	57590	E0F6
Costanti	58077	E2DD
Costanti	58171	E33B
Inizial. vettori RAM	58232	E378
Subroutine per pagina 0	58247	E387
Inizializzazione Basic	58276	E3A4
Inizializzazione vettori	58191	E44F

A partire dalla locazione 58247 (E387H) si ritrova una copia della routine, che è in pagina zero a partire dal byte 115 (0073H); essa può essere usata per ripristinare la routine di pagina zero nella sua versione originale, qualora sia stata modificata.

CAPITOLO 8

ASSEMBLER E LINGUAGGIO MACCHINA

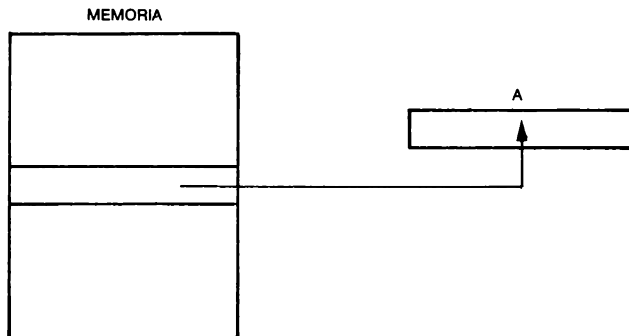
8.1 LINGUAGGIO MACCHINA DEL 6502

Per "linguaggio macchina" si intende l'insieme delle configurazioni binarie (codici operativi) che il calcolatore è in grado di capire ed eseguire (set di istruzioni); ognuna di esse rappresenta un'operazione eseguita via hardware dalla CPU. Il linguaggio macchina di un calcolatore dipende dalla CPU che esso possiede, che, nel caso del VIC è il 6502.
Il linguaggio macchina è in binario; è intuibile come scrivere un programma in questo linguaggio sia lungo, laborioso e difficile. D'altra parte i programmi in questo linguaggio consentono una velocità operativa maggiore e un controllo sulla macchina molto più diretto che in BASIC, ed in certi casi è quindi necessario sobbarcarsi questa fatica, quando l'applicazione lo richiede.
Esistono diverse possibilità per scrivere programmi in linguaggio macchina, utilizzando invece che un codice numerico, un codice simbolico, in cui ogni istruzione viene rappresentata da una parola che ricorda, in qualche modo, l'operazione che l'istruzione stessa esegue. Ad esempio l'istruzione di somma si esprime con ADD. Un codice di questo tipo si chiama mnemonico; il linguaggio costituito da questi codici si chiama assembler. Siccome però il linguaggio comprensibile direttamente al calcolatore resta comunque quello binario, occorre un programma che effettui la traduzione dall'assembler al linguaggio macchina, che prende il nome di assemblatore.
In questo capitolo si spiega come scrivere ed eseguire programmi in linguaggio macchina sul VIC, introducendoli in esadecimale, in decimale o in assembler. Non c'è la possibilità di scrivere in binario, per fortuna...

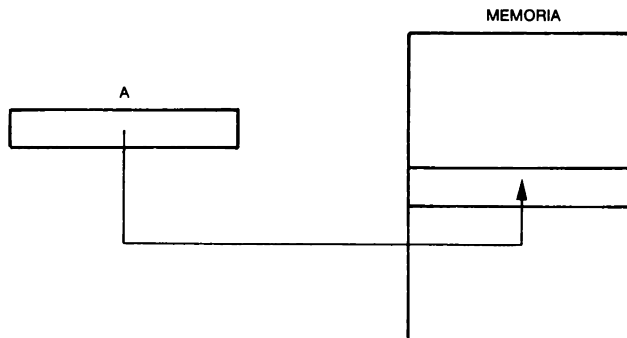
Il set delle istruzioni macchina del 6502 è composto dai seguenti gruppi:
1) trasferimento dati
2) operazioni sullo stack
3) aritmetico-logiche
4) rotazione e scorrimento
5) confronto
6) salto
7) chiamata e ritorno da subroutine
8) controllo

1)Trasferimento dati

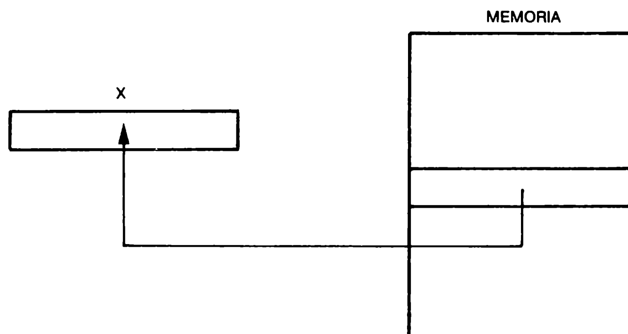
LDA (Load Accumulator)
Il dato specificato nell'operando viene caricato nell'accumulatore.



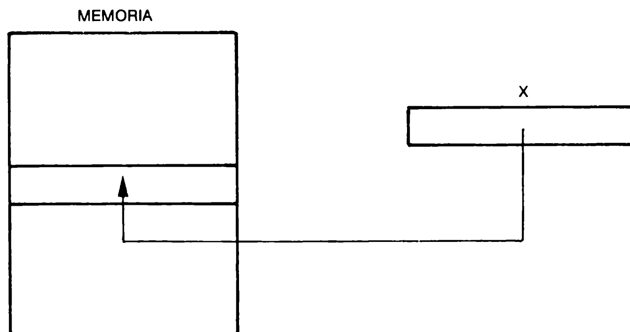
STA (Store Accumulator)
Copia nella memoria, all'indirizzo specificato, il contenuto dell'accumulatore.



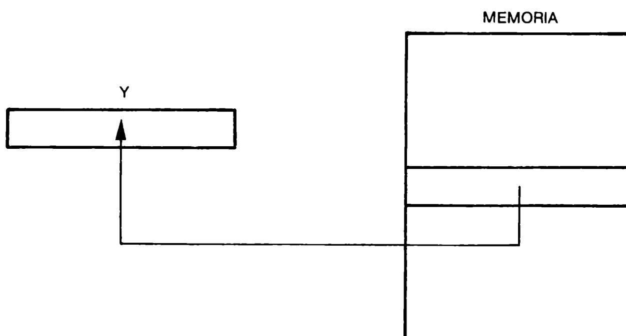
LDX (Load X)
Il dato specificato dall'operando viene caricato nel registro X.



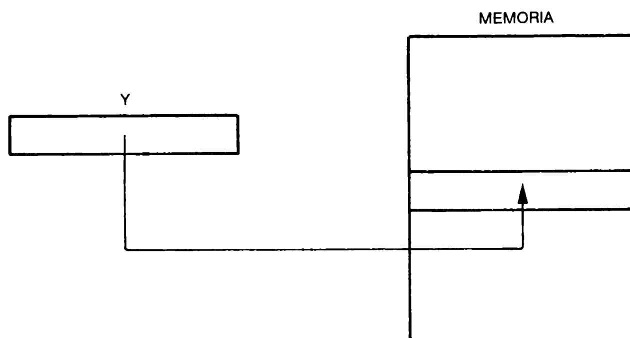
STX (Store X)
Copia nella memoria, all'indirizzo specificato, il contenuto del registro X.



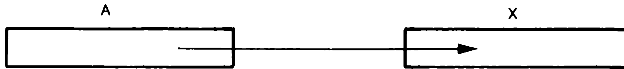
LDY (Load Y)
Il dato specificato nell'operando viene caricato nel registro Y.



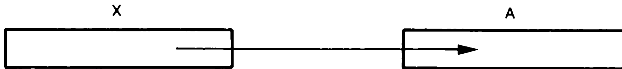
STY (Store Y)
Copia nella memoria, all'indirizzo specificato, il contenuto del registro Y.



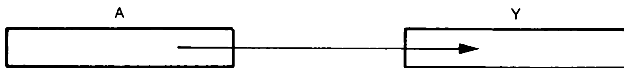
TAX (Transfer Accumulator into X)
Copia il contenuto dell'accumulatore nel registro X.



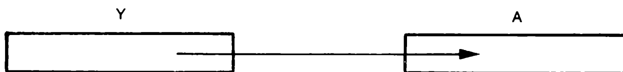
TXA (Transfer X into Accumulator)
Copia il contenuto del registro X nell'accumulatore.



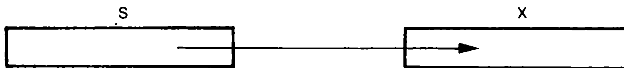
TAY (Transfer Accumulator into Y)
Copia il contenuto dell'accumulatore nel registro Y.



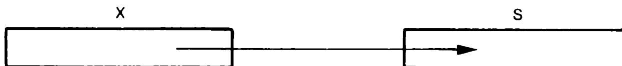
TYA (Transfer Y into A)
Copia il contenuto del registro Y nell'accumulatore.



TSX (Transfer S into X)
Copia il contenuto dello Stack Pointer nel registro X.



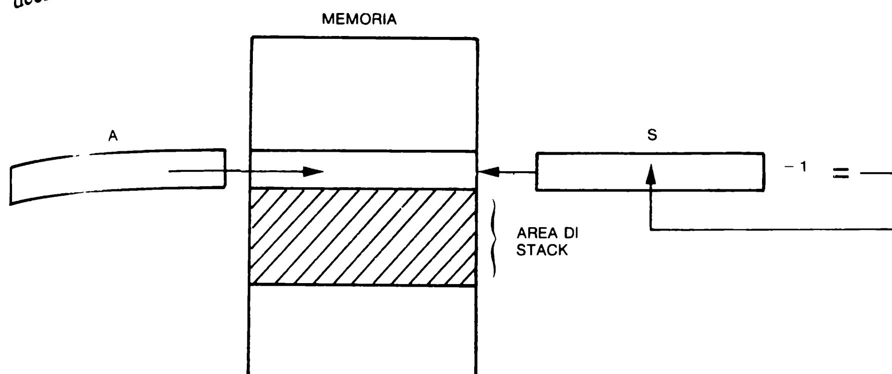
TXS (Transfer X into S)
Copia il contenuto del registro X nello Stack Pointer.



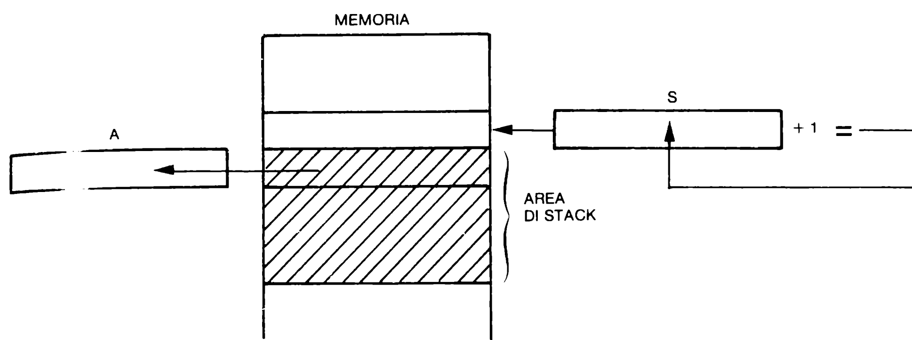
2)Operazioni sullo Stack

PHA (Push Accumulator)
Il contenuto dell'accumulatore viene inserito nell'area di stack, nella prima locazio-

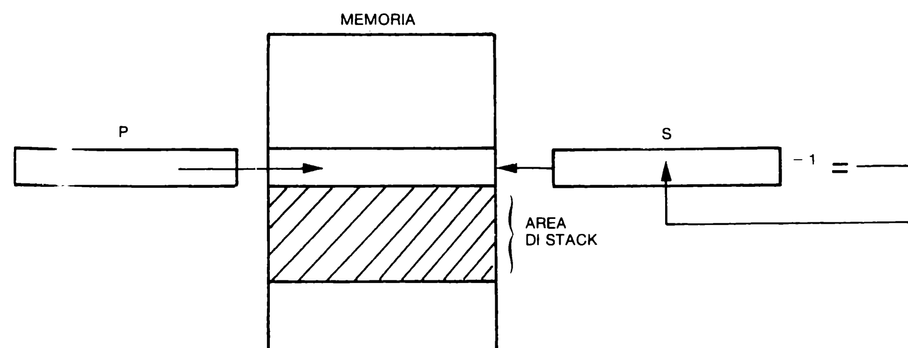
ne libera, il cui indirizzo si trova nello stack pointer; lo stack pointer viene poi decrementato di 1.



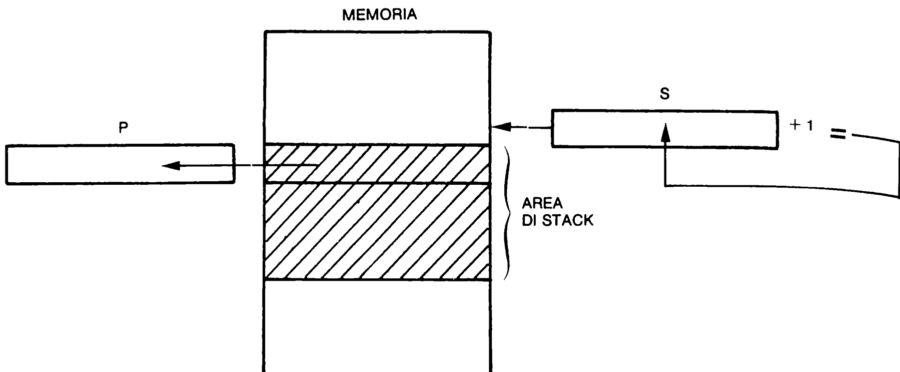
PLA (Pull Accumulator)
Il contenuto del byte che si trova in cima all'area di stack viene copiato nell'accumulatore, e il puntatore viene incrementato di 1.



PHP (Push P)
Il contenuto del registro di stato P viene inserito nell'area di stack; il puntatore dell'area di stack viene decrementato di 1.



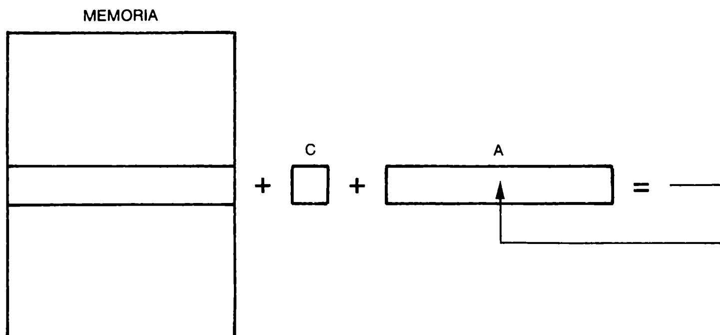
PLP (Pull P)
Il contenuto del byte che si trova in cima all'area di stack viene copiato nel registro di stato, e il puntatore viene incrementato di 1.



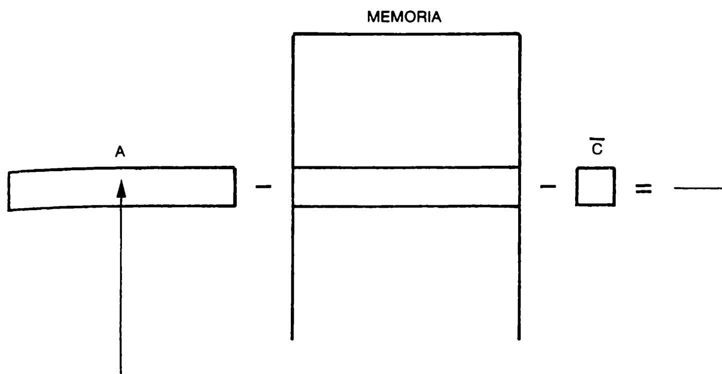
3)Aritmetico-logiche

Si ricorda che le operazioni aritmetiche vengono eseguite su numeri binari o BCD a seconda che il flag D sia a 0 o a 1.

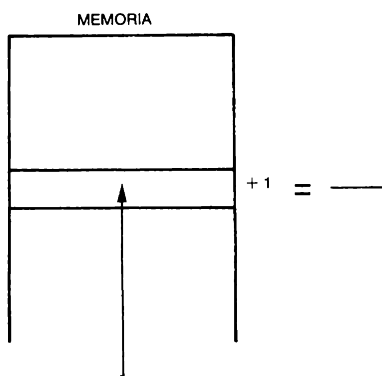
ADC
Somma il contenuto dell'accumulatore all'operando, aggiunge il bit del carry e pone il risultato nell'accumulatore.



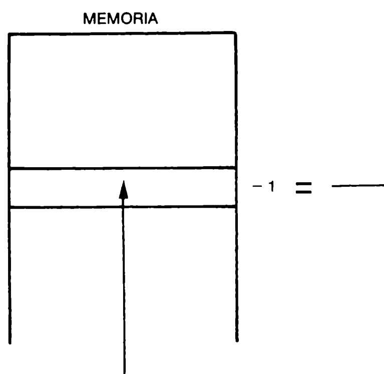
SBC
Somma all'operando il carry complementato e sottrae il risultato dall'accumulatore; il risultato resta nell'accumulatore.



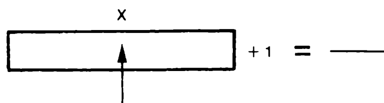
INC
Incrementa l'operando di 1



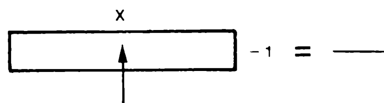
DEC
Decrementa di 1 l'operando.



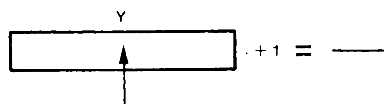
INX
Incrementa di 1 il registro X.



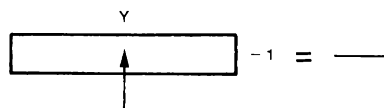
DEX
Decrementa di 1 il registro X.



INY
Incrementa di 1 il registro Y.

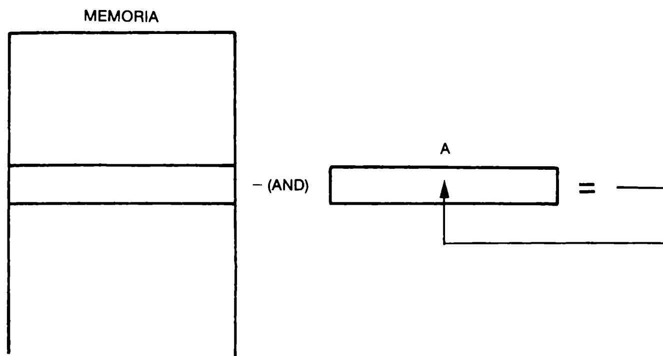


DEY
Decrementa di 1 il registro Y.



AND
Esegue l'AND bit a bit del contenuto dell'accumulatore con l'operando, e pone il risultato nell'accumulatore.
Si ricorda che l'AND tra 2 bit dà i seguenti risultati:

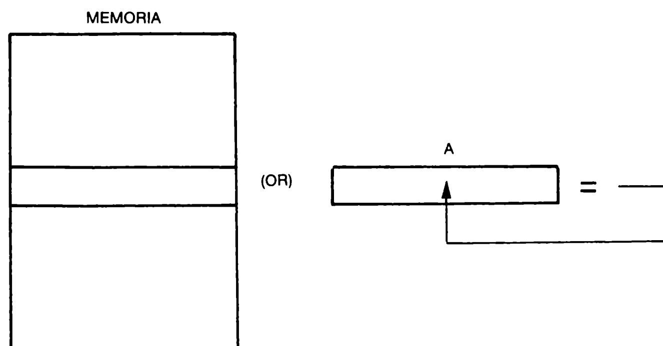
A	B	A(AND)B
0	0	0
0	1	0
1	0	0
1	1	1



ORA

Esegue l'OR bit a bit del contenuto dell'accumulatore con l'operando; il risultato è caricato nell'accumulatore. Il risultato dell'OR tra due bit è illustrato di seguito.

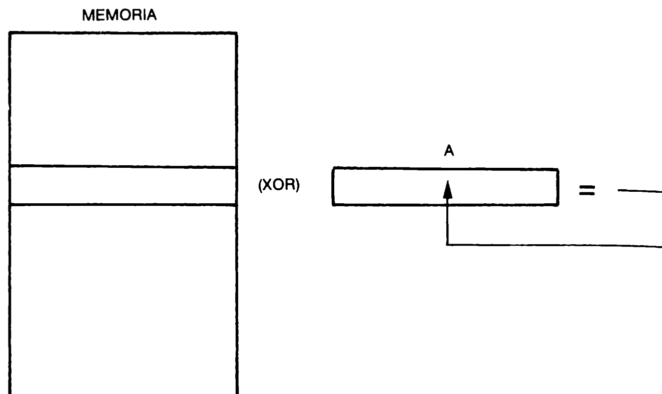
A	B	A(OR)B
0	0	0
0	1	1
1	0	1
1	1	1



EOR

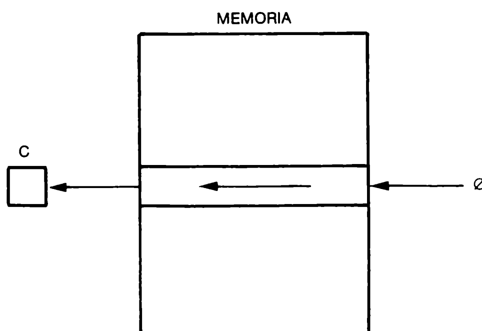
Esegue l'OR-Esclusivo dell'operando con il contenuto dell'accumulatore. I risultati dell'OR-Esclusivo tra due bit sono riportati di seguito.

A	B	A(XOR)B
0	0	0
0	1	1
1	0	1
1	1	0

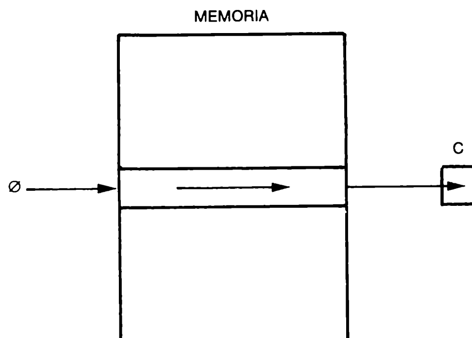


4)Rotazione e scorrimento

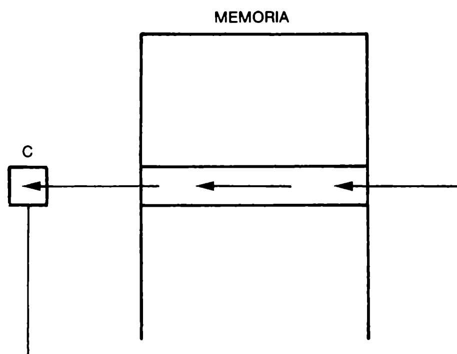
ASL (Arithmetic Shift Left)
Sposta di un bit a sinistra il contenuto dell'operando, inserendo uno 0 a destra e copiando nel carry il bit 7.



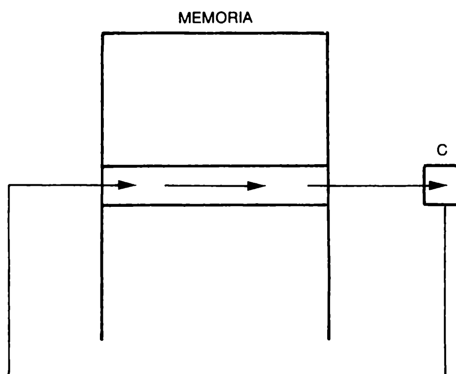
LSR (Logical Shift Right)
Sposta di un bit a destra l'operando, inserendo uno 0 da sinistra e copiando il bit 0 nel carry.



ROL (Rotate Left)
Il contenuto dell'operando viene ruotato verso sinistra di un bit; il contenuto del carry precedente l'operazione viene inserito nel bit 0 e il bit 7 viene copiato nel carry.

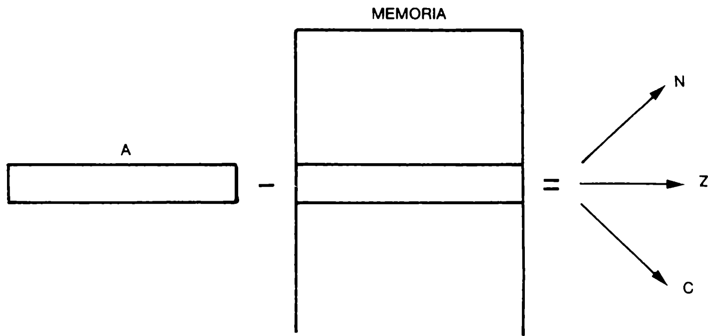


ROR (Rotate Right)
Il contenuto dell'operando viene ruotato di un bit verso destra; il contenuto del carry viene copiato nel bit 7 e il bit 0 viene copiato nel carry.

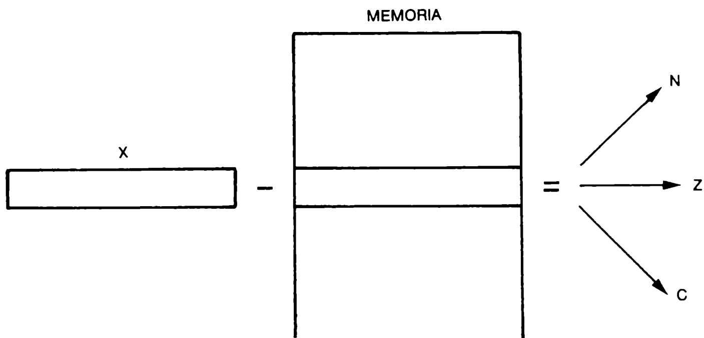


5)Confronto

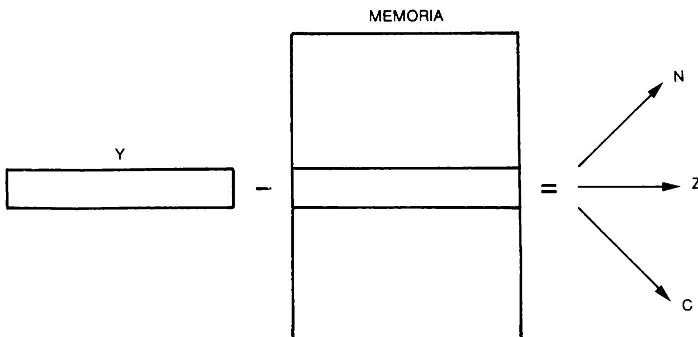
CMP (Compare)
Esegue il confronto tra l'operando e il contenuto dell'accumulatore, senza modificare nè l'uno nè l'altro; il confronto viene eseguito facendo un'ideale sottrazione tra il contenuto dell'accumulatore e l'operando. In base all'esito del confronto vengono modificati i flag N, Z e C.



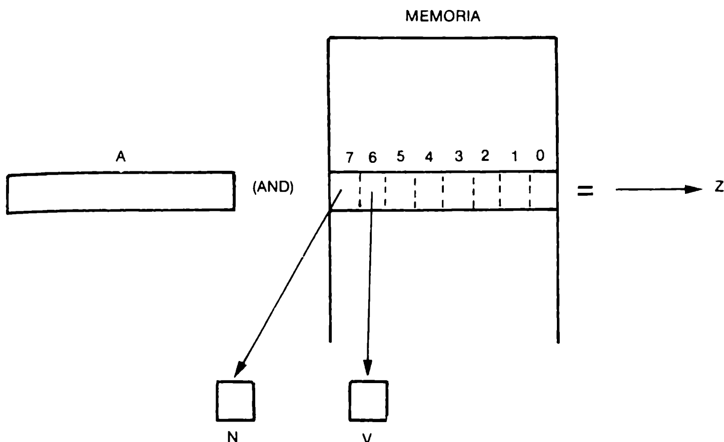
CPX (Compare to X)
Come la precedente, con il registro X al posto dell'accumulatore.



CPY (Compare to Y)
Come la precedente, con il registro Y al posto di X.



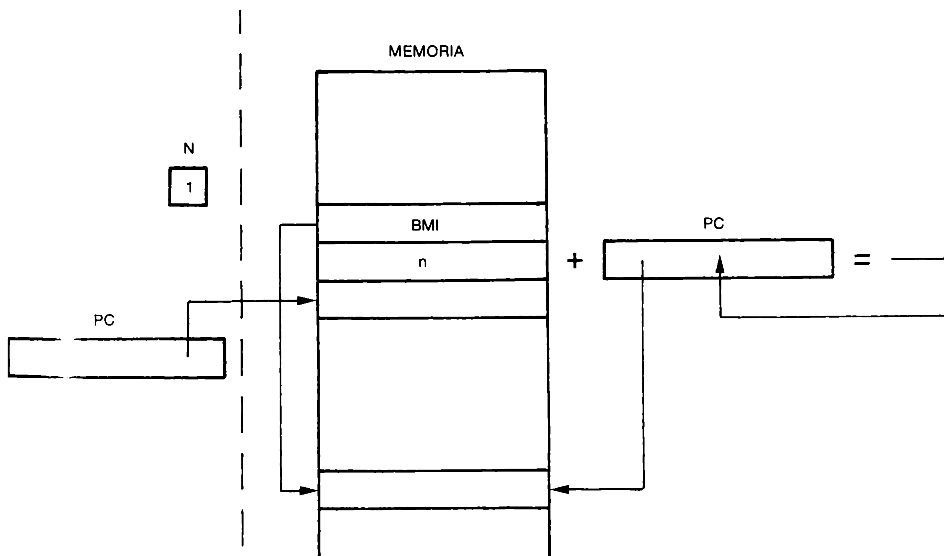
BIT
Esegue l'AND tra l'operando e il contenuto dell'accumulatore, lasciando però invariati sia l'uno che l'altro. Il flag Z viene messo a 1 se il risultato è 0, altrimenti a 0. I bit 6 e 7 dell'operando sono copiati rispettivamente nei flag V e N del registro di stato.



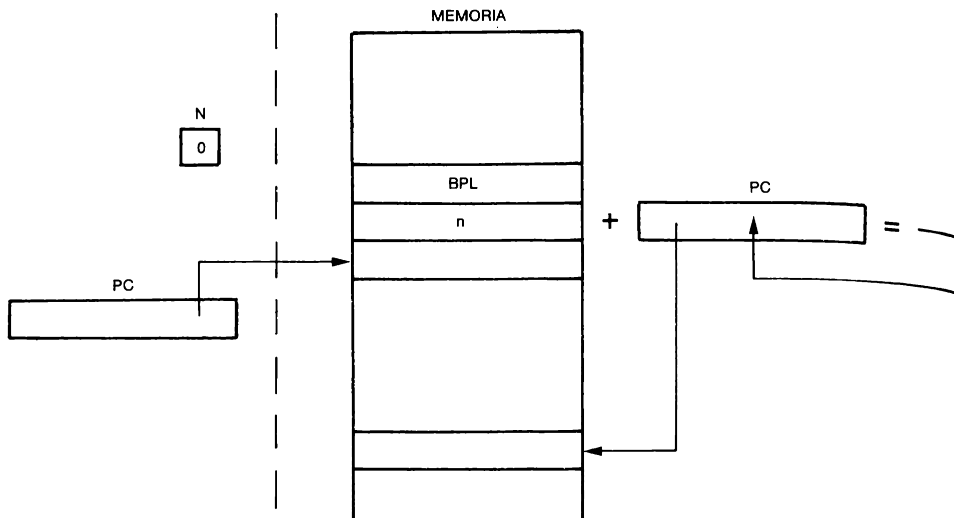
6)Salto

Tutte le istruzioni di BRANCH (diramazione) usano il modo di indirizzamento relativo; l'operando (spiazzamento) è il numero di byte da saltare, calcolato partendo dall'indirizzo dell'istruzione successiva a quella di salto. Il numero è positivo se il salto va fatto in avanti, negativo in complemento a 2 se il salto è all'indietro. Considerando i numeri rappresentabili in 8 bit in complemento a 2, è possibile saltare fino a 128 byte all'indietro e 127 in avanti.

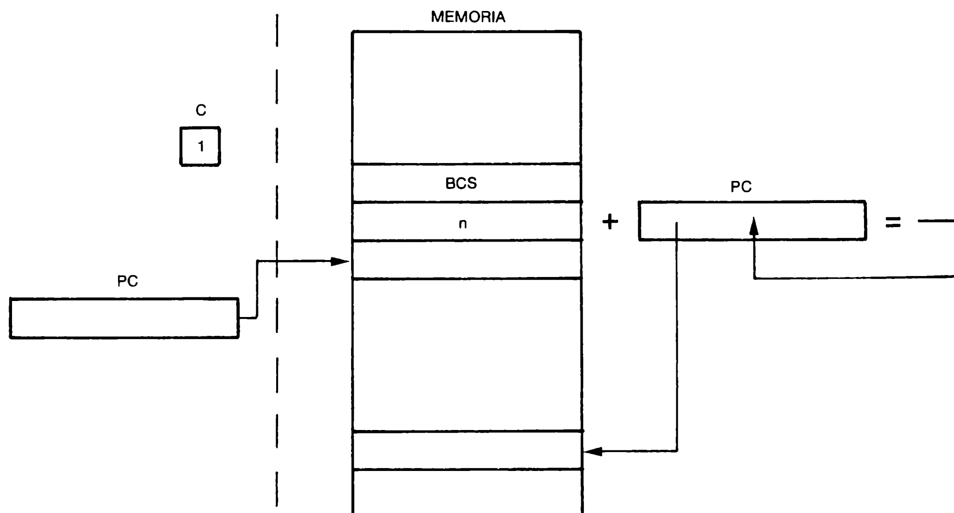
BMI (Branch on Minus)
Trasferisce il controllo all'indirizzo specificato, se il flag N è uguale a 1, cioè se il risultato dell'ultima operazione che lo ha modificato è negativo.



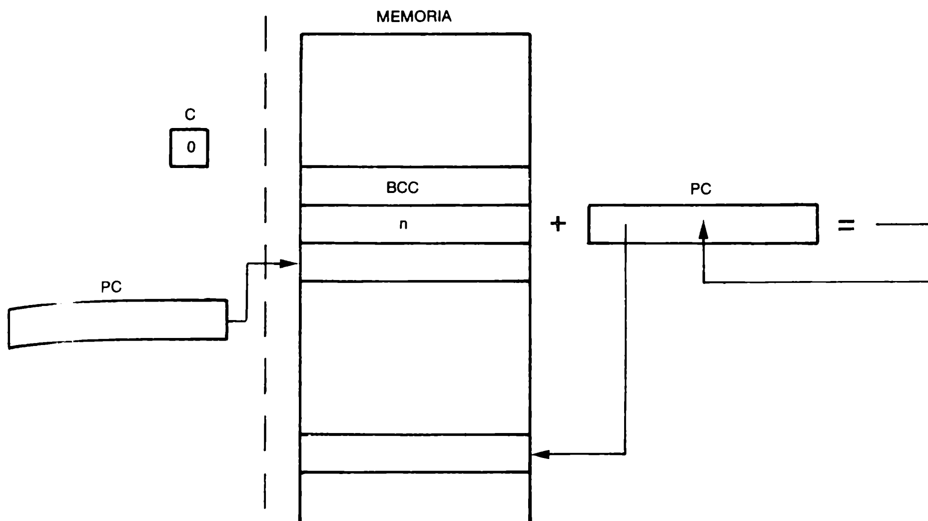
BPL (Branch on Plus)
Come la precedente se il flag N è a 0, cioè se il risultato dell'ultima operazione che lo ha modificato è positivo.



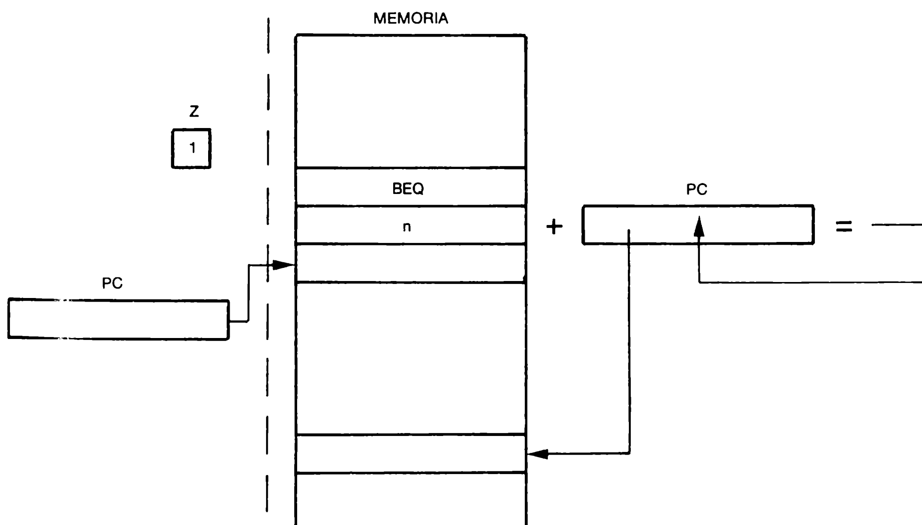
BCS (Branch on Carry Set)
Come la precedente, nel caso che il carry sia a 1.



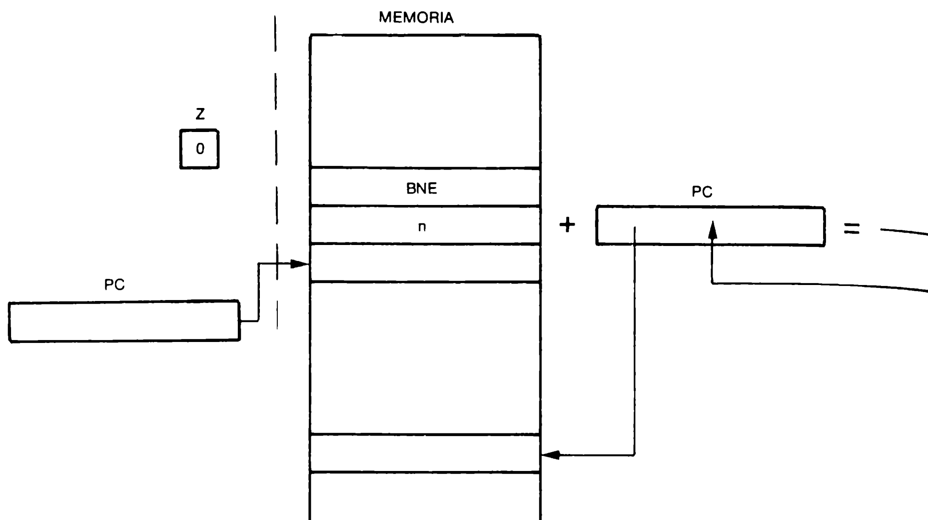
BCC (Branch on Carry Clear)
Come la precedente se il carry è a 0.



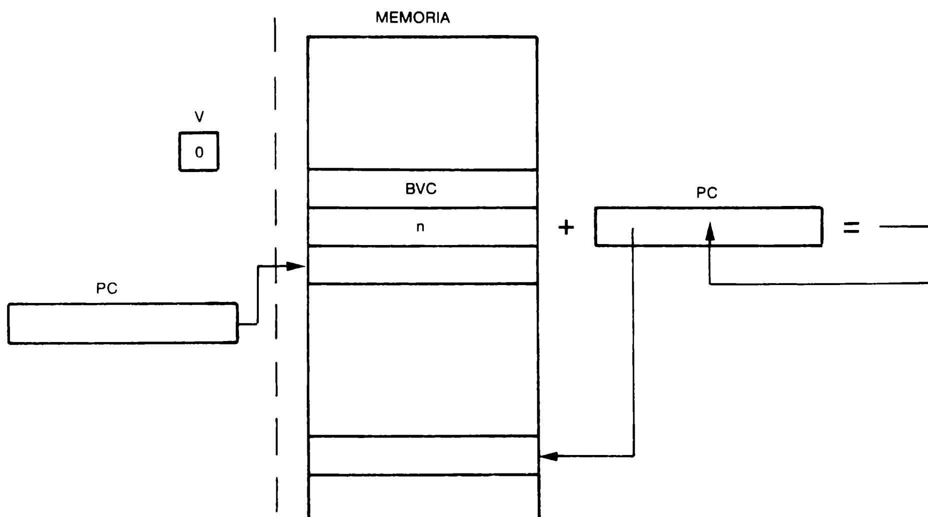
BEQ (Branch if Equal to Zero)
Come la precedente, se il flag Z è a 1, cioè se l'ultima operazione che lo ha modificato ha dato risultato 0.



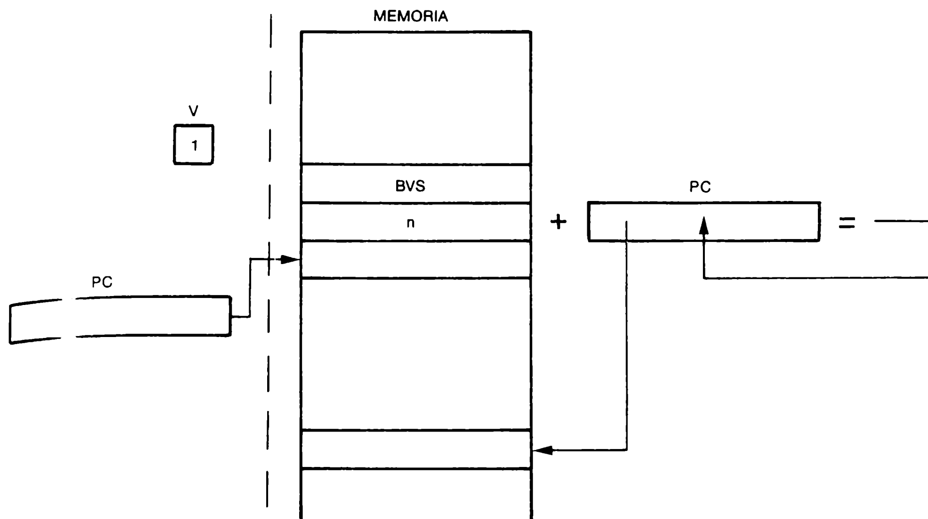
BNE (Branch on Not Equal to zero)
Come la precedente se il flag Z è a 0, cioè se il risultato dell'ultima operazione che lo ha modificato è diverso da 0.



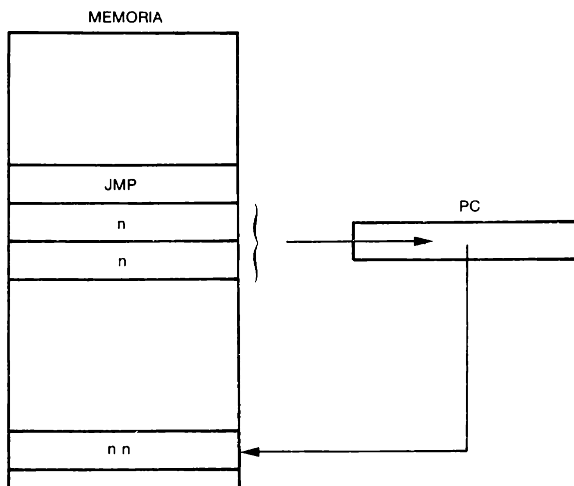
BVC (Branch on Overflow Clear)
Come la precedente se il flag V è a 0, cioè se l'ultima operazione che lo ha modificato ha dato un riporto dal bit 6 del risultato. Si ricorda che il flag di overflow segnala eventuali errori nelle operazioni su numeri con segno.



BVS (Branch on Overflow Set)
Come la precedente se il flag V è a 1.



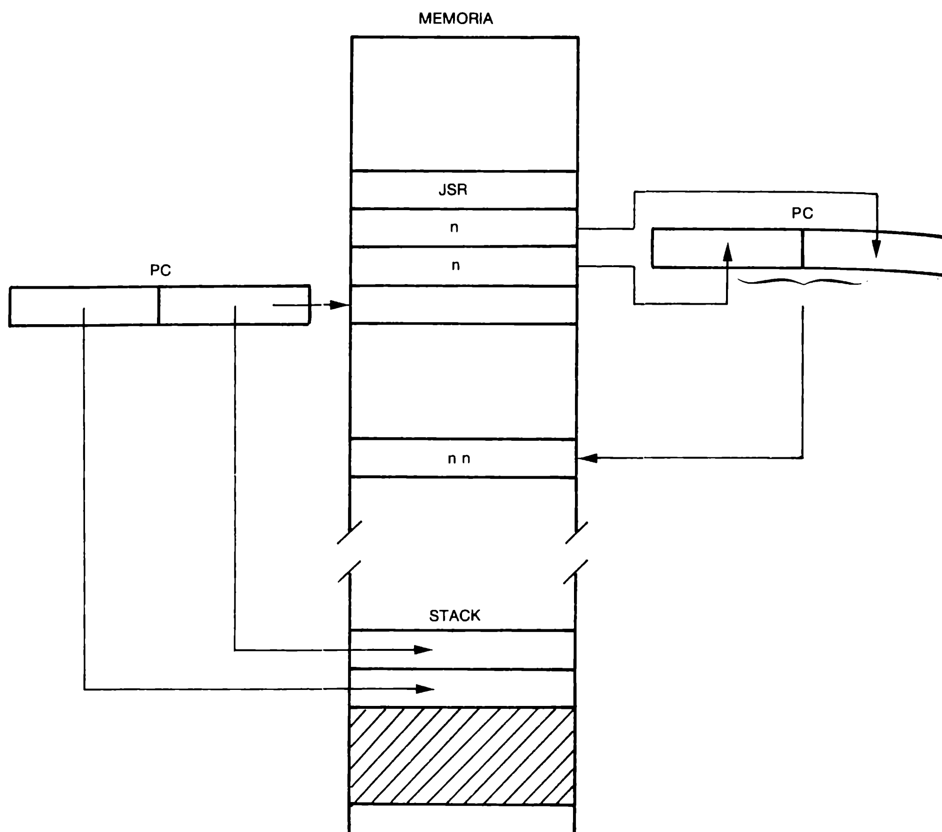
JMP (Jump)
L'indirizzo specificato viene caricato nel Program Counter, provocando quindi il trasferimento del controllo all'indirizzo stesso.



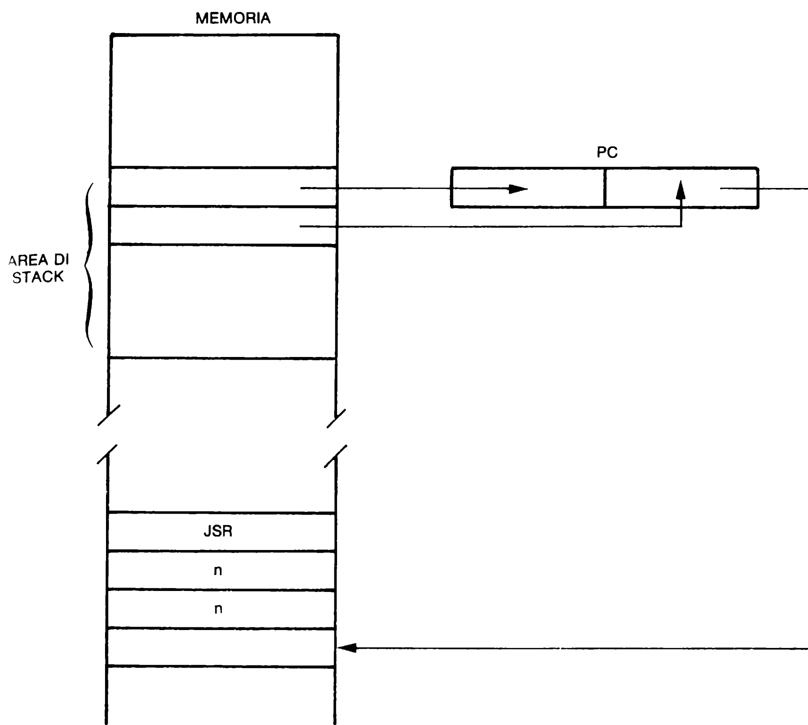
7) Chiamata e ritorno da subroutine

JSR (Jump to Subroutine)
Il contenuto del Program Counter, decrementato di 1, viene salvato nell'area di

stack, e nel Program Counter viene caricato l'indirizzo specificato; con ciò il controllo viene trasferito al sottoprogramma che inizia a quell'indirizzo. Questa istruzione prende il nome di "chiamata a subroutine".

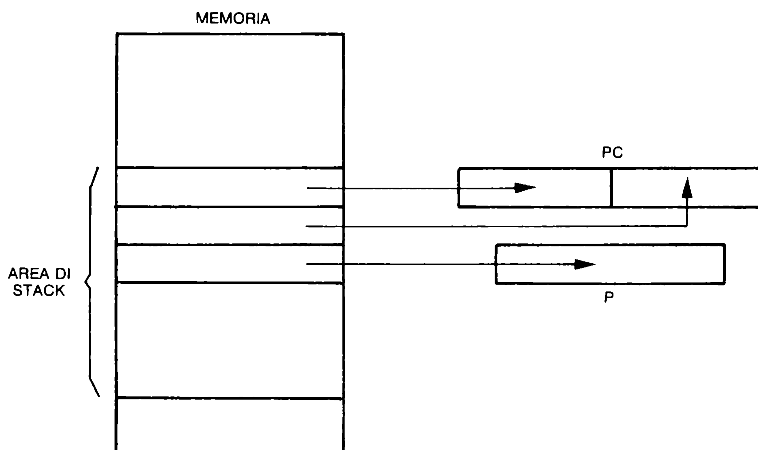


RTS (Return from Subroutine)
Copia i due byte in cima all'area di stack rispettivamente nel byte meno significativo e in quello più significativo del Program Counter; aggiorna il puntatore dell'area di stack e incrementa il Program Counter di 1.



RTI (Return from Interrupt)

Deve essere l'ultima istruzione di qualsiasi routine di servizio di un interrupt. Preleva i primi tre byte dell'area di stack e li carica rispettivamente nel registro di stato, nel byte meno significativo e in quello più significativo del Program Counter; il puntatore dell'area di stack viene aggiornato.



8)Controllo

CLC (Clear Carry)
Azzera il flag del carry.

SEC (Set Carry flag)
Pone a 1 il flag del carry.

CLD (Clear Decimal flag)
Azzera il flag per l'aritmetica decimale (D); le operazioni aritmetiche che seguono questa istruzione, fino a un'altra che modifichi lo stesso flag, sono eseguite su numeri binari.

SED (Set Decimal flag)
Pone a 1 il flag D; le operazioni aritmetiche vengono eseguite su numeri BCD.

CLI (Clear Interrupt flag)
Azzera il flag I di disabilitazione degli interrupt; in questo modo gli interrupt risultano abilitati.

SEI (Set Interrupt flag)
Pone a 1 il flag di disabilitazione degli interrupt; gli interrupt vengono quindi disabilitati.

CLV (Clear Overflow flag)
Azzera il flag di overflow V.

NOP (No Operation)
La CPU non esegue alcuna operazione, oltra quella di incrementare il Program Counter, e passa all'istruzione successiva.

BRK
Provoca un'interruzione del programma, e il salto all'indirizzo contenuto nelle locazioni 65534 e 65535 di memoria. Prima di eseguire il salto, il contenuto del PC e del registro di stato vengono conservati nell'area di stack, e vengono ripristinati al

termine dell'esecuzione della routine di servizio dell'interruzione. Il flag B del registro di stato, prima di essere conservato, viene messo a 1, il che consente di riconoscere un'interruzione generata via hardware sulla linea IRQ da una generata dalla BRK. Il PC viene incrementato di 2, prima di essere copiato nell'area di stack. La BRK si usa in genere per inserire degli arresti temporanei nell'esecuzione di un programma, allo scopo di verificarne il funzionamento; viene quindi provvisoriamente sostituita a un'istruzione del programma. Se questa è a 3 byte, la BRK deve essere seguita da 2 NOP, e il rientro dalla routine di interrupt avviene correttamente. Se l'istruzione sostituita è a 1 o a 2 byte, è opportuno che la stessa routine di interrupt modifichi l'indirizzo di rientro, decrementandolo rispettivamente di 1 o di 2.

8.2 VICMON E IL SUO ASSEMBLER

VICMON è un cartridge che contiene un monitor, studiato per facilitare la programmazione in linguaggio macchina sul VIC.
Esso fornisce le seguenti funzioni:
-assemblaggio e disassemblaggio di programmi;
-visualizzazione e modifiche del contenuto della memoria;
-spostamento di blocchi di byte da una zona a un'altra di memoria;
-visualizzazione e modifiche del contenuto dei registri del 6502;
-inserimento di stop provvisori in programmi in linguaggio macchina;
-esecuzione con diverse modalità di programmi in linguaggio macchina;
-caricamento e salvataggio di programmi da e su periferiche.

Il cartridge VICMON va inserito a calcolatore spento; una volta acceso il calcolatore, per lanciare il monitor occorre digitare

SYS 24576 oppure

SYS 6*4096

seguito dal tasto RETURN.
Sul video comparirà un'immagine del tipo

```
B*
    PC   SR AC XR YR SP
.;  603E 33 00 63 00 F7
```

A questo punto il monitor è in grado di accettare comandi.
Sotto VICMON, il "prompt" è rappresentato da un "." (punto).

Eventuali errori nell'introduzione di un comando vengono segnalati con un "?" immediatamente dopo la posizione dell'errore. Per correggerlo basta riscrivere il comando.
Con VICMON è possibile, tra l'altro, scrivere programmi in assembler, invece che in esadecimale.
Nelle tabelle seguenti sono elencate in ordine alfabetico le istruzioni assembler di VICMON; per ognuna di esse sono forniti il codice mnemonico, i modi di indirizzamento, il codice esadecimale, il numero di byte occupati, il numero di cicli macchina impiegati, e infine le eventuali modifiche dei flag.

Le notazioni usate sono:

A	accumulatore
X,Y	registri indice
M	memoria
P	registro di stato
S	stack pointer
m	modificato
/	non modificato
+	somma
∧	AND
−	sottrazione
∀	XOR
↑	trasferimento dall'area di stack
↓	trasferimento nell'area di stack
→	trasferimento
←	trasferimento
∨	OR
PC	Program Counter
PCH	Program Counter High
PCL	Program Counter Low
op	operando
#	modo di indirizzamento immediato

Le lettere distintive dei flag sono:

N = flag del segno
Z = flag di zero
C = flag del carry
I = flag di disabilitazione degli interrupt
D = flag di aritmetica decimale
V = flag di overflow

ADC

A+M+C→A,C

N Z C I D V
m m m / / m

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	ADC #op	69	2	2
pag zero	ADC op	65	2	3
pag zero, X	ADC op,X	75	2	4
ass	ADC op	6D	3	4
ass,X	ADC op,X	7D	3	4*
ass,Y	ADC op,Y	79	3	4*
(ind X)	ADC (op,X)	61	2	6
(ind),Y	ADC (op),Y	71	2	5*

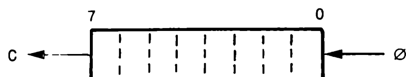
AND

A ∧ M → A

N Z C I D V
m m / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	AND #op	29	2	2
pag zero	AND op	25	2	3
pag zero,X	AND op,X	35	2	4
ass	AND op	2D	3	4
ass X	AND op,X	3D	3	4*
ass Y	AND op,Y	39	3	4*
(ind,X)	AND (op,X)	21	2	6
(ind),Y	AND (op),Y	31	2	5

ASL



N Z C I D V
m m m / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
acc	ASL A	0A	1	2
pag zero	ASL op	06	2	5
pag zero,X	ASL op,X	16	2	6
ass	ASL op	0E	3	6
ass X	ASL op,X	1E	3	7

BCC
Salta se C=0

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
rel	BCC op	90	2	2*

BCS
Salta se C=1

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
rel	BCS op	B0	2	2*

BEQ
Salta se Z=1

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
rel	BEQ op	F0	2	2*

BIT
A ∧ M, M7→N, M6→V

N Z C I D V
M7 n / / / M6

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
pag zero	BIT op	24	2	3
ass	BIT op	2C	3	4

BMI
Salta se N=1

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
rel	BMI op	30	2	2*

BNE
Salta se Z=0

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
rel	BNE op	D0	2	2*

BPL
Salta se N=0

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
rel	BPL op	10	2	2*

BRK
Interrupt; PC+2↓P↓

N Z C I D V
/ / / 1 / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	BRK	00	1	7

BVC
Salta se V=0

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
rel	BVC op	50	2	2*

BVS
Salta se V=1

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
rel	BVS op	70	2	2*

CLC
0→C

N Z C I D V
/ / 0 / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	CLC	18	1	2

CLD
0→D

N Z C I D V
/ / / / 0 /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	CLD	D8	1	2

CLI
0→I

N Z C I D V
/ / / 0 / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	CLI	58	1	2

CLV
0→V

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	CLV	B8	1	2

CMP
A-M

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	CMP#op	C9	2	2
pag zero	CMP op	C5	2	3
pag zero,X	CMP op,X	D5	2	4
ass	CMP op	CD	3	4
ass X	CMP op,X	DD	3	4*
ass,Y	CMP op,Y	D9	3	4*
(ind,X)	CMP (op,X)	C1	2	6
(ind),Y	CMP (op),Y	D1	2	5*

CPX
X-M

N Z C I D V
m m m / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	CPX #op	E0	2	2
pag zero	CPX op	E4	2	3
ass	CPX op	EC	3	4

CPY
Y-M

N Z C I D V
m m m / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	CPY #op	C0	2	2
pag zero	CPY op	C4	2	3
ass	CPY op	CC	3	4

DEC
M-1→M

N Z C I D V
m m / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
pag zero	DEC op	C6	2	5
pag zero,X	DEC op,X	D6	2	6
ass	DEC op	CE	3	6
ass,X	DEC op,X	DE	3	7

DEX
X-1→X

N Z C I D V
m m / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	DEX CA	1	2	

DEY
Y-1→Y

N Z C I D V
m m / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	DEY	88	1	2

EOR
A∀M→A

N Z C I D V
m m / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	EOR #op	49	2	2
pag zero	EOR op	45	2	3
pag zero,X	EOR op,X	55	2	4
ass	EOR op	4D	3	4
ass,X	EOR op,X	5D	3	4*
ass,Y	EOR op,Y	59	3	4*
(ind,X)	EOR (op,X)	41	2	6
(ind),Y	EOR (op),Y	51	2	5*

INC
M+1→M

N Z C I D V
m m / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
pag zero	INC op	E6	2	5
pag zero,X	INC op,X	F6	2	6
ass	INC op	EE	3	6
ass,X	INC op,X	FE	3	7

INX
X+1→X

N Z C I D V
m m / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	INX	E8	1	2

INY
Y+1→Y

N Z C I D V
m m / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	INY	C8	1	2

JMP
(PC+1)→PCL
(PC+2)→PCH

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
ass	JMP op	4C	3	3
ind	JMP (op)	6C	3	5

JSR
PC+1 ↓, (PC+1)→PCL
(PC+2)→PCH

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
ass	JSR op	20	3	6

LDA
M→A

N Z C I D V
m m / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	LDA #op	A9	2	2
pag zero	LDA op	A5	2	3
pag zero,X	LDA op,X	B5	2	4
ass	LDA op	AD	3	4
ass,X	LDA op,X	BD	3	4*
ass,Y	LDA op,Y	B9	3	4*
(ind,X)	LDA (op,X)	A1	2	6
(ind),Y	LDA (op),Y	B1	2	5*

LDX
M→X

N Z C I D V
m m / / / /

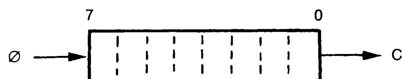
MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	LDX #op	A2	2	2
pag zero	LDX op	A6	2	3
pag zero,Y	LDX op,Y	B6	2	4
ass	LDX op	AE	3	4
ass,Y	LDX op,Y	BE	3	4*

LDY
M→Y

N Z C I D V
m m / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	LDY #op	A0	2	2
pag zero	LDY op	A4	2	3
pag zero,X	LDY op,X	B4	2	4
ass	LDY op	AC	3	4
ass,X	LDY op,X	BC	3	4*

LSR



N Z C I D V
0 m m / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
acc	LSR A	4A	1	2
pag zero	LSR op	46	2	5
pag zero,X	LSR op,X	56	2	6
ass	LSR op	4E	3	6
ass,X	LSR op,X	5E	3	7

NOP
Nessuna operazione

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	NOP	EA	1	2

ORA
A∨M→A

N Z C I D V
m m / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	ORA #op	09	2	2
pag zero	ORA op	05	2	3
pag zero,X	ORA op,X	15	2	4
ass	ORA op	0D	3	4
ass,X	ORA op,X	1D	3	4*
ass,Y	ORA op,Y	19	3	4*
(ind,X)	ORA (op,X)	01	2	6
(ind),Y	ORA (op),Y	11	2	5

PHA
A

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	PHA	48	1	3

PHP
P↓

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	PHP	08	1	3

PLA
A↑

N Z C I D V
m m / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	PLA	68	1	4

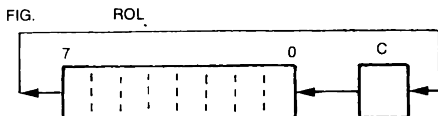
PLP
P↑

N Z C I D V
dallo stack

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	PLP	28	1	4

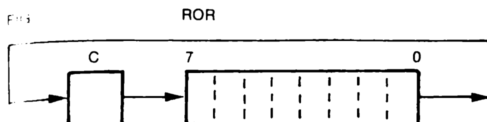
ROL

FIG. ROL



N Z C I D V
m m m / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
acc	ROL A	2A	1	2
pag zero	ROL op	26	2	5
pag zero,X	ROL op,X	36	2	6
ass	ROL op	2E	3	6
ass,X	ROL op,X	3E	3	7



ROR

N Z C I D V
m m m / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
acc	ROR A	6A	1	2
pag zero	ROR op	66	2	5
pag zero,X	ROR op,X	76	2	6
ass	ROR op	6E	3	6
ass,X	ROR,op	7E	3	7

RTI
P↑ PC↑

N Z C I D V
dallo stack

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	RTI	40	1	6

RTS
PC↑, PC+1→PC

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	RTS	60	1	6

SBC
A-M-C→A
C=riporto

N Z C I D V
m m m / / m

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
imm	SBC #op	E 9	2	2
pag zero	SBC op	E5	2	3
pag zero,X	SBC op,X	F5	2	4
ass	SBC op	ED	3	4
ass,X	SBC op,X	FD	3	4*
ass,Y	SBC op,Y	F9	3	4*
(ind,X)	SBC (op,X)	E1	2	6
(ind),Y	SBC (op),Y	F1	2	5*

SEC
1→C

N Z C I D V
/ / 1 / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	SEC	38	1	2

SED
1→D

N Z C I D V
/ / / / 1 /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	SED	F8	1	2

SEI
1→I

N Z C I D V
/ / / 1 / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	SEI	78	1	2

STA
A→M

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
pag zero	STA op	85	2	3
pag zero,X	STA op,X	95	2	4
ass	STA op	8D	3	4
ass,X	STA op,X	9D	3	5
ass,Y	STA op,Y	99	3	5
(ind,X)	STA (op,X)	81	2	6
(ind),Y	STA (op),Y	91	2	6

STX
X→M

NZCIDV
//////

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
pag zero	STX op	86	2	3
pag zero,Y	STX op,Y	96	2	4
ass	STX op	8E	3	4

STY
Y→M

NZCIDV
//////

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
pag zero	STY op	84	2	3
pag zero,X	STY op,X	94	2	4
ass	STY op	8C	3	4

TAX
A→X

NZCIDV
mm////

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	TAX	AA	1	2

TAY
A→Y

NZCIDV
mm////

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	TAY	A8	1	2

TYA
Y→A

NZCIDV
mm////

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	TYA	98	1	2

TSX
S→X

N Z C I D V
m m / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	TSX	BA	1	2

TXA
X→A

N Z C I D V
m m / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	TXA	8A	1	2

TXS
X→S

N Z C I D V
/ / / / / /

MODO INDIRIZZAMENTO	CODICE SIMBOLICO	CODICE ESADEC	n BYTE	n CICLI
impl	TXS	9A	1	2

* NOTA: il numero di cicli aumenta se si esce dalla pagina corrente.

Si elencano di seguito brevemente i comandi di VICMON, rimandando per un approfondimento sul loro utilizzo al Manuale venduto insieme con il cartridge. Le notazioni usate sono:

(ind) :un indirizzo in esadecimale a 2 byte (es.: 0400)
(dev) :un numero di dispositivo, esadecimale, a 1 byte (es.: 08)
(opcod) :un codice mnemonico assembler (es.:LDA)
(operando) :un operando compatibile con il codice mnemonico specificato
(val) :un numero esadecimale a 1 byte (es.: FF)
(dati) :una stringa alfanumerica racchiusa tra virgolette
(rif) :un indirizzo esadecimale a 2 byte (es.: 2000)
(offset) :un indirizzo esadecimale a 2 byte (es.: 3000)

A (ind), (opcod), (operando)
Consente di introdurre una linea di programma in assembler, specificandone l'indirizzo; l'istruzione viene tradotta in codice macchina e memorizzata.

B (ind) o B (ind),n
Consente di inserire un breakpoint (arresto parziale) in un certo indirizzo di memoria; il programma si ferma prima di eseguire l'istruzione che si trova a quell'indirizzo. Se l'indirizzo si trova all'interno di un loop, è possibile specificare dopo quante volte il programma si deve arrestare. L'esecuzione interrotta da un breakpoint può essere ripresa con uno dei comandi di lancio del programma che si vedranno in seguito.

D (ind) o D (ind),(ind)
Consente di disassemblare una o più istruzioni in linguaggio macchina che si trovano memorizzate in una certa zona di memoria; le istruzioni specificate vengono riconvertite da binario in asssembler.

E (ind)
Consente di creare una "pagina zero" virtuale di 256 byte in una zona di memoria prefissata; serve per evitare che le variabili usate dal programma in linguaggio macchina e dallo stesso VICMON in pagina zero vadano a sovrapporsi a quelle usate dal sistema nella stessa pagina.

F (ind), (ind), (val)
Consente di scrivere un valore noto in un blocco di byte successivi della memoria.

G o G (ind)
Lancia il programma a partire dalla locazione specificata.

H (ind), (ind), (dati)
Cerca un dato o una stringa di caratteri all'interno di un blocco di memoria; visualizza sul video il dato e gli indirizzi a cui è stato trovato.

I (ind),(ind) o I (ind)
Visualizza sul video in campo inverso i caratteri stampabili i cui codici si trovano in un certo blocco di memoria.

J
In un'esecuzione single step (un'istruzione alla volta) del programma, consente di lanciare una particolare subroutine ed eseguirla alla velocità normale; si usa associato al comando W, che serve per eseguire i comandi single step.

L "nome-file", (dev)
Carica in memoria da disco o da nastro un file specificato.

M (ind), (ind)   o M (ind)
Visualizza in esadecimale il contenuto dei byte di un blocco di memoria.

N (ind), (ind), (offset), (estremo inf), (estremo sup), W
Modifica opportunamente gli indirizzi assoluti presenti in un programma quando questo viene rilocato , con il comando T, in una zona di memoria diversa da quella per cui era stato scritto; in pratica somma algebricamente un valore di offset a tutti gli indirizzi.

Q   o Q (ind)
Lancia un programma a partire da un indirizzo specificato; quando incontra un breakpoint, o quando viene premuto da tastiera STOP seguito da X, passa al modo single step, come se fosse stato dato un comando W.

R
Visualizza il contenuto dei registri della CPU in esadecimale.

RB
Elimina un breakpoint fissato in precedenza.

S"nome file", (dev), (ind), (ind)
Ricopia il contenuto di una area di memoria su nastro o su disco, assegnandogli un nome di file.

T (ind), (ind), (ind)
Trasferisce il contenuto di una zona di memoria in un'altra.

W   o W (ind)
Predispone l'esecuzione single step di un programma, a partire da un certo indirizzo; esegue l'istruzione che si trova a quell'indirizzo e si ferma; se si preme lo spazio, esegue l'istruzione successiva, e così via.

X
Restituisce il controllo al BASIC.

Segue un esempio di programma scritto con VICMON. Si è seguita la seguente procedura:

1)Si è scritto il programma PROVA MONITOR in BASIC, e lo si è fatto girare, ottenendo di spostare il TOP della memoria (registri 643 e 644) a 7552 (1D80H), con la SYS 58232. Si osservi sul video il numero di byte disponibili: sarà minore del solito.

```
10 REM PROVA MONITOR
20 POKE643,128:POKE644,29:SYS58232:STOP

READY.
```

2) Si è scritto il programma P2MONITOR, che non è altro che la routine HARDCOPY, già vista, con le righe iniziali di lancio.

```
10 REM P2MONITOR
20 GOSUB60000
30 STOP
60000 REM CRT COPY
60010 G1$=CHR$(145)
60020 OPEN4,4:PRINT#4:G1=7658
60030 FORG0=0TO22:G0$=G1$:G1=G1+22
60040 FORG2=G1TOG1+21:G3=PEEK(G2)
60050 IFG3>128THENG3=G3-128:G4=1:G0$=G0$+CHR$(18)
60060 IF(G3>0)*(G3<32)THENG3=G3+64:GOTO60100
60070 IF(G3>31)*(G3<64)THEN60100
60080 IF(G3>63)*(G3<96)THENG3=G3+128:GOTO60100
60090 IF(G3>95)*(G3<128)THENG3=G3+64:GOTO60100
60100 G0$=G0$+CHR$(G3)
60110 IFG4=1THENG0$=G0$+CHR$(146):G4=0
60120 NEXTG2:PRINT#4,G0$:NEXTG0
60130 PRINT#4:CLOSE4
60140 RETURN
```

3) si è lanciato VICMON con SYS 24576, e si è richiesta la funzione A per scrivere il programma in assembler a partire da 1D80H ; con la X si è usciti da VICMON, e si è eseguito il programma BASIC presente in memoria, ottenendo l'hard copy del video sotto riportato.

```
SYS24576

B*
   PC  SR AC XR YR SP
.;603E 33 00 63 00 F6
.A 1D80 SEI
.A 1D81 LDA #$90
.A 1D83 STA $0314
.A 1D86 LDA #$1D
.A 1D88 STA $0315
.A 1D8B LDA #$FF
.A 1D8D STA $AF
.A 1D8F RTS
.A 1D90
.X
READY.
RUN
```

4 Si è lanciato nuovamente VICMON con SYS 24576, e si è usata la funzione M per

leggere la memoria da 1D80H a 1D8FH; con la X si è tornati al BASIC e si è eseguito il programma di HARDCOPY ottenendo il contenuto di memoria.

```
SYS24576

B*
   PC  SR AC XR YR SP
.;603E 33 00 63 00 F6
.M1D80,1D8F

.:1D80 78 A9 90 8D 14
.:1D85 03 A9 1D 8D 15
.:1D8A 03 A9 FF 85 AF
.:1D8F 60 02 08 08 30
..1D94
.X
READY.
RUN
```

In 1D80H iniziano i codici del programma caricato in assembler e assemblato dall'assemblatore di VICMON.
Se si spegne il calcolatore, alla successiva accensione il TOP della memoria torna ai valori normali.

8.3 PROGRAMMAZIONE IN LINGUAGGIO MACCHINA

VICMON è uno strumento potente e flessibile per scrivere ed eseguire programmi in assembler o in linguaggio macchina, soprattutto se sono programmi stand-alone, che cioè vengono eseguiti da soli,e non in ambiente BASIC.
Una delle caratteristiche più interessanti del VIC è però quella di poter eseguire programmi in linguaggio macchina all'interno di programmi BASIC, consentendo così una struttura particolarmente agile e flessibile al programma, che può sfruttare le caratteristiche più vantaggiose dei due linguaggi, a seconda delle operazioni da compiere.
Per poter lavorare sotto BASIC e contemporaneamente accedere a programmi in codice macchina esistono diverse possibilità:

- le parti in linguaggio macchina vengono scritte sotto VICMON, e sono poi salvate insieme con il programma BASIC che le richiama, ad esempio salvando tutto il contenuto della memoria utente, dalla locazione 400H (1024). Questo sistema non è molto comodo; conviene usare VICMON per grossi programmi in linguaggio macchina stand-alone, mentre per inserire in un programma BASIC routine brevi in linguaggio macchina conviene usare uno dei metodi seguenti.

- il programma in codice macchina viene scritto all'inizio del programma BASIC con una serie di REM, suddiviso in blocchi lunghi al massimo 80 byte. Durante l'introduzione del programma BASIC, le istruzioni REM vengono riempite di

caratteri “dummy” (fittizi), che servono solo a riservare lo spazio per il codice macchina; poi viene utilizzato VICMON per trovare in memoria la posizione di queste REM, e per scrivere nelle locazioni ad esse riservate i codici macchina. Non è una procedura semplice; l'ideale è che una routine in linguaggio macchina entri in una sola REM (80 byte), così non si hanno problemi per saltare i 6 byte che si hanno tra i diversi “corpi” delle REM, dovuti al byte 0 di fine istruzione precedente, ai 2 byte di numero linea, ai 2 byte di link, e al byte del “token” REM. Inoltre è semplice trovare l'inizio di ogni routine, subito dopo il token REM.
Quando si lista un programma così composto, nelle REM compariranno una serie di caratteri senza senso, che sono l'equivalente ASCII dei codici macchina che vi sono stati scritti.

O il programma in codice macchina viene inserito nel programma BASIC con un semplice loader, che utilizza le POKE per scrivere i codici macchina decimali direttamente in memoria, come è mostrato nell'esempio del paragrafo 8.4.
Il programma seguente, invece, è un loader esadecimale. Esso chiede inizialmente l'indirizzo da cui deve essere caricato il programma, e poi, uno alla volta, i byte esadecimali del codice macchina, seguiti ognuno da RETURN. Introducendo un “*” si segnala la fine del programma. Di ogni byte il loader controlla che sia al massimo di due caratteri esadecimali (0-9, A-F); in caso contrario segnala errore. Se il byte è formalmente corretto viene scritto in memoria all'indirizzo che gli compete dopo averlo convertito in decimale.

```
1 REM CARICATORE ESADECIMALE
10 PRINT"▮▮▮▮IND. DEC. PRIMO BYTE";:INPUTN
20 PRINT"CONFERMI  S/N ";:INPUTR$
30 IFR$<>"S"THEN10
40 PRINT"SCRIVI I BYTE DA CARI-CARE IN ESADECIMALE  UNO PER VOLTA"
50 PRINT"SCRIVI * PER TERMINARE"
60 M=N
65 PRINT"▮BYTE ESADECIMALI"
70 INPUTA$:IFA$="*"THENPRINT "FINITO CARICAMENTO AL BYTE ";M-1:STOP
80 IFLEN(A$)<1ORLEN(A$)>2THEN200
90 A=ASC(A$):B=ASC(RIGHT$(A$,1))
100 A=A-48:B=B-48
110 IFA<0THEN200
120 IFA>9THENA=A-7:IFA<10THEN200
130 IFA>15THEN200
140 IFB<0THEN200
150 IFB>9THENB=B-7:IFB<10THEN200
160 IFB>15THEN200
170 X=A*16+B
180 POKEM,X:M=M+1
190 GOTO70
200 PRINT"ERRORE RISCRIVI":GOTO70
```

Per richiamare una routine in linguaggio macchina da un programma BASIC esistono due istruzioni:

● USR(X), che trasferisce il controllo all'indirizzo che si trova nelle locazioni 1 e 2 di memoria, definito opportunamente dall'utente. La variabile X è un parametro che può essere scambiato tra programma principale e routine, e che si trova, alla fine, nel "floating accumulator n.1", nella locazione 61, e nella variabile dell'utente, a cui viene assegnato il valore calcolato da USR.

● SYS(X), che trasferisce il controllo alla locazione X, dove X è una variabile o una costante che contiene il valore decimale dell'indirizzo di partenza della routine; eventuali parametri possono essere passati dal programma chiamante alla subroutine e viceversa, tramite istruzioni POKE e PEEK in indirizzi assoluti.

Un altro modo per eseguire programmi in linguaggio macchina consiste nell'aggiungerli o sostituirli a routine del sistema operativo, con una delle seguenti modalità:

● se il codice macchina è scritto su ROM, a partire dall'indirizzo 40960 (A000H), al momento dell'accensione il VIC dà il controllo a questo programma, invece che all'interprete BASIC; sfruttando questa caratteristica si possono aggiungere programmi al BASIC, oppure modificare alcune funzioni di I/O, o, addirittura, mandare in esecuzione, al posto del BASIC e del sistema operativo, del software particolare; questo è quello che accade quando si inseriscono i cartridge, che sono preparati per occupare il blocco ROM che parte dall'indirizzo A000H (si veda paragrafo 7.1).
Quando si accende il calcolatore, viene eseguita una sequenza di inizializzazione, che si trova in ROM, alla locazione 64802 (FD22H). Questa per prima cosa controlla se nello spazio degli indirizzi che inizia alla locazione 40960 (A000H) è inserita della ROM. Questo controllo viene effettuato cercando la stringa di caratteri "AOCBM" nelle locazioni dalla 40964 (A004H) in avanti; se questa viene trovata, il controllo passa alla routine il cui indirizzo di partenza si trova nelle locazioni 40960 (A000H) e 40961 (A001H), che esegue l'inizializzazione necessaria per il software contenuto nella ROM aggiunta. Altrimenti il controllo passa alla locazione 64815 (FD2FH), dove si trova la routine che predispone il calcolatore a lavorare in BASIC.

● il codice macchina può essere aggiunto alle routine che hanno l'indirizzo memorizzato in un vettore in RAM, da 788 a 819 (0314H-0333H), oppure da 768 a 777 (0300H-0309H), modificando la tabella degli indirizzi. Tra queste c'è la routine di servizio degli interrupt mascherabili (IRQ), che viene lanciata 60 volte al secondo dalla porta VIA collegata al piedino IRQ della CPU; se ad essa si aggiungono parti scritte dall 'utente, è possibile eseguire sistematicamente operazioni diverse da quelle previste, o aggiunte a quelle previste. Nel vettore della tabella in RAM si deve scrivere l'indirizzo di partenza del programma in codice macchina che si vuole aggiungere; esso deve terminare con un salto all'indirizzo di partenza della relativa routine di servizio standard.

● infine il codice macchina dell'utente può essere inserito nella routine CHARGOT, in pagina zero, per aggiungere comandi extra all'interprete BASIC (si veda il paragrafo 7.2).
In tutti i casi, per tornare alla situazione normale, occorre ripristinare i valori standard degli indirizzi e il codice standard della CHARGOT.

Ed ora, infine, alcuni suggerimenti per chi programma in linguaggio macchina.
La posizione migliore per un programma in codice macchina è nella zona più alta della memoria utente; questa viene utilizzata dal BASIC per memorizzarvi le stringhe. Il BASIC conserva in due puntatori in pagina zero (locazioni 51 e 52, 55 e 56) rispettivamente, l'indirizzo più alto della memoria utente e l'estremo superiore della zona riservata alle stringhe. Per evitare che il codice macchina ed eventuali stringhe si sovrappongono, basta abbassare il valore di questi puntatori, riservando così uno spazio ignorato dal BASIC, e completamente a disposizione del programma in codice macchina.
Le seguenti istruzioni

```
POKE 51,0: POKE 52,16
POKE 55,0: POKE 56,16
```

mostrano un esempio in cui, ai puntatori di prima, viene dato il valore 4096. Per ripristinare il valore iniziale, si usa l'istruzione CLR. Se però si vuole utilizzare la CLR senza spostare i puntatori, si deve scrivere l'indirizzo di fine della memoria nei byte 643 e 644, e richiamare la routine del sistema che determina la memoria libera per il BASIC, in base ai contenuti di questi due byte, scrivendo SYS 58232; in questo caso, per ripristinare la situazione normale occorre spegnere e riaccendere il calcolatore, o ripetere la procedura al contrario. Dopo avere usato la SYS si vedrà che il messaggio iniziale sui byte liberi è cambiato rispetto alla norma. Una procedura analoga si può usare per riservare spazio all'inizio dell'area BASIC, modificando i valori dei puntatori 642 e 643, e chiamando ancora la SYS 58232.
Le variabili usate dal programma in codice macchina possono essere in qualsiasi zona della memoria RAM, ovviamente non occupata dalle variabili di sistema o dal programma; per ottenere però la massima velocità di esecuzione e la minima occupazione di memoria, è opportuno memorizzarle in pagina zero. Se il programma è stand alone, cioè non gira insieme a un programma BASIC, si può usare tutta la parte della pagina zero normalmente occupata dalle variabili dell'interprete BASIC (locazioni 0-43);altrimenti sono disponibili solo le locazioni lasciate libere dal BASIC e dal sistema operativo. Se queste non bastano, si può rilocare la pagina zero di sistema in un'altra parte della memoria, prima di eseguire il programma in linguaggio macchina, e ripristinarla dopo. VICMON, coma si è visto, ha un comando particolare per questa operazione; se non si usa VICMON occorre scrivere un breve programma ad hoc.
Quando il programma in codice macchina è richiamato con una USR o con una SYS, occorre salvare il contenuto dei registri prima della sua esecuzione, e ripristi-

narli prima di ritornare al programma BASIC; per far ciò basta che all'inizio della routine i registri vengano conservati nell'area di stack, e prelevati immediatamente prima della fine.
Si ricorda infine che gli indirizzi a 2 byte che compaiono nelle istruzioni del programma in codice macchina devono essere scritti in due byte successivi, di cui il primo contiene la parte bassa dell'indirizzo (byte meno significativo), e il secondo la parte alta (byte più significativo).

8.4 ESEMPIO DI MODIFICA DELLA ROUTINE DI INTERRUPT

La routine di servizio dell'interrupt IRQ viene chiamata 60 volte al secondo, e svolge diverse funzioni:

.Incrementa la variabile TI e di conseguenza la TI$ del BASIC.

.Controlla se sulla tastiera è stato premuto un tasto e memorizza il carattere corrispondente al tasto premuto.

.Controlla i tasti del registratore: se un tasto viene rilasciato, allora viene ripristinata l'alimentazione del registratore.

.Produce il lampeggiamento del cursore del video.

Il meccanismo di servizio è il seguente:

.Giunge alla CPU il segnale di interrupt.

.La CPU legge il contenuto delle locazioni di memoria 788 e 789 (0314H-0315H) in RAM e salta all'indirizzo contenuto in esse.

.Quando viene riconosciuta un'istruzione RTI (ritorno da interrupt) la CPU riprende ad eseguire il programma interrotto.

L'utente può intercettare la routine di interrupt tramite le locazioni sopra citate e fargli fare altre operazioni oltre a quelle standard. Si propone un programma esempio nel quale, intercettando la routine di interrupt, si fa suonare al VIC una nota musichetta, mentre svolge altri programmi.

Segue la routine in assembler che inizializza i puntatori in 788 e 789 (0314H-0315H) all'indirizzo 7568 (1D90H) e setta il byte 175 (AFH) con 8 bit 1, come flag della routine (tale byte è usato dal sistema in fase di lettura programmi da cassetta).

Programma assembler

Ind dec.	Ind.esad.	Istruzioni	Significato
7552	1D80	SEI	disabilita gli interrupt
7553	1D81	LDA #90	carica in accumulatore il numero 144 (90H)
7555	1D83	STA 0314	memorizza nel byte 788 (0314H) il numero
7558	1D86	LDA #1D	carica in accumulatore il numero 29 (1DH)
7560	1D88	STA 0315	memorizza nel byte 789 (0315H) il numero
7563	1D8B	LDA #FF	carica in accumulatore 128 (FFH)
7565	1D8D	STA AF	memorizza nel byte 175 (AFH) il numero
7567	1D8F	RTS	esce dal sottoprogramma

Si ricordi che per caricare un indirizzo in due byte si deve separare la parte più significativa dalla meno significativa, dividendo per 256 l'indirizzo decimale; la parte meno significativa (resto della divisione precedente) va nel primo byte e la più significativa nel secondo.

La routine assembler precedente, che occupa 16 byte, dà luogo ai seguenti valori numerici:

in decimale: 120, 169, 144, 141, 20, 3, 169, 29, 141, 21, 3, 169, 255, 133, 175, 96

e in esadecimale: 78, A9, 90, 8D, 14, 03, A9, 1D, 8D, 15, 03, A9, FF, 85, AF, 60

essi verranno caricati con dei DATA nel programma BASIC e poi scritti con le istruzioni POKE nelle locazioni desiderate.

Segue la routine musicale:

Ind.dec.	Ind.esad.	Istruzioni	Significato
7568	1D90	LDA AF	carica il flag 175 (AFH) in accumulatore
7570	1D92	BEQ 1DAE	salta se zero a 7598 (1DAEH)
7572	1D94	DEC B0	decrementa del contenuto del byte 176 (B0H)
7574	1D96	BNE 1DAE	salta se non zero a 7598 (1DAEH)
7576	1D98	LDA #13	carica in accumulatore il numero 19 (13H)
7578	1D9A	STA B0	memorizza il numero nel byte 176 (B0H)
7580	1D9C	LDY B1	carica nel registro Y il contenuto del byte 177 (B1H) (1 all'inizio)
7582	1D9E	INC B1	aggiunge 1 a 177 (B1H)
7584	1DA0	CPY #30	confronta Y con 48
7586	1DA2	BNE 1DA8	salta se non zero a 7592 (1DA8H)
7588	1DA4	LDA #01	carica in accumulatore il numero 1 (1H)
7590	1DA6	STA B1	memorizza il numero nel byte 177 (B1H)
7592	1DA8	LDA 1DCF,Y	carica in acc. il contenuto di 1DCFH +Y (frequenza nota)
7595	1DAB	STA 900A	memorizza nel byte 36874 (900AH) la nota
7598	1DAE	JMP EABF	salta all'indirizzo 60095 (EABFH) indirizzo della normale routine di interrupt in 60095 (EABFH)

La routine, che occupa 33 byte, dà luogo ai seguenti valori numerici:

in decimale: 165, 175, 240, 26, 198, 176, 208, 22, 169, 19, 133, 176, 164, 177, 230, 177, 192, 48, 208, 4, 169, 1, 133, 177, 185, 207, 29, 141, 10, 144, 76, 191, 234

e in esadecimale: A5, AF, F0, 1A, C6, B0, D0, 16, A9, 13, 85, B0, A4, B1, E6, B1, C0, 30, D0, 04, A9, 01, 85, B1, B9, CF, 1D, 8D, 0A, 90, 4C, BF, EA

Anche questi byte vengono caricati con dei DATA nel programma BASIC.

Il byte 176 (B0H) fa da temporizzatore della nota, la quale dura 19 chiamate di

interrupt (circa 0.3 sec.). Il byte 177 (B1H) contiene (con l'aggiunta del contenuto del registro Y) il puntatore alla nota corrente; le note sono 48. Le note vengono poste nel generatore di suono, byte 36874 (900AH) e il programma BASIC pone poi il volume nel byte 36878.

Le note devono essere poste nei byte che vanno da 7632 a 7679 (1DD0H-1DFFH); esse sono in esadecimale:

CF D9 DF E3 E4 E3 DF D9 CF D9 DF E3 E4 E3 DF D9
B7 C7 CF D4 D7 D4 CF C7 CF D9 DF E3 E4 E3 DF D9
BF CB D4 CB B7 C7 CF C7 CF D9 DB DD DF BF DF BF

e corrispondono in decimale a:

207 217 223 227 228 227 223 217
207 217 223 227 228 227 223 217
183 199 207 212 215 212 207 199
207 217 223 227 228 227 223 217
191 203 212 203 183 199 207 199
207 217 219 221 223 191 223 191

Segue il listato del programma Basic.

```
20 FORI=0TO48:READA:POKE7552+I,A:NEXT
30 FORI=0TO47:READA:POKE7632+I,A:NEXT
40 POKE36878,15:POKE176,19:POKE177,1:REM***INIZZIALIZZA I CONTATORI***
45 POKE643,128:POKE644,29:SYS7552:SYS58232
50 REM****************    ON : PEEK(175)<>0     OFF: PEEK(175)=0   *************
99 REM ******************* DATI PROGRAMMA ******************
100 DATA120,169,144,141,20,3,169,29,141,21,3,169,255,133,175,96,165,175,240,26
110 DATA198,176,208,22,169,19,133,176,164,177,230,177,192,48,208,4,169,1,133,177
120 DATA 185,207,29,141,10,144,76,191,234
199 REM******************** DATI MUSICA ********************
200 DATA207,217,223,227,228,227,223,217,207,217,223,227,228,227,223,217,183,199
210 DATA207,212,215,212,207,199,207,217,223,227,228,227,223,217,191,203,212,203
220 DATA 183,199,207,199,207,217,219,221,223,191,223,191
```

Nella linea 20 vengono caricati a partire dall'indirizzo 7552 i 49 byte delle due routine in linguaggio macchina. Nella linea 30 vengono caricate le 48 note a partire dal byte 7632. Nella linea 40 viene posto a 15 il byte 36878 del volume (solo la parte volume del suono), e vengono inizializzati i due contatori usati dal programma in linguaggio macchina: il byte 176 (B0H) con il numero 19 per la durata della nota, e il byte 177 (B1H) con il numero 1 (viene usato per puntare alle note usando anche il registro Y). Nella linea 45 con le istruzioni POKE vengono modificati i contenuti

dei 2 byte 643 e 644 che danno l'indirizzo di fine della memoria; essi vengono posti all'indirizzo 7552 (1D80H) (1 (29*256+128=7552). In tale modo il programma in linguaggio macchina che si trova a quell'indirizzo non viene disturbato. Inoltre con SYS7552 viene lanciato il programma in linguaggio macchina che parte da 7552; la prima routine. Con SYS58232 (E378H) viene lanciata la routine del sistema operativo che fa comparire sul video il messaggio con la versione del Basic e i K di memoria disponibili. Si vedrà che sono un po' meno del solito a causa dello spostamento del TOP della memoria. Il programma BASIC ha a questo punto terminato il suo lavoro; il VIC però suona una musichetta mentre si può lavorare in BASIC. E' chiaro che appena si comincia a lavorare il programma sparisce, ma il codice macchina resta e continua a lavorare fino a quando non si spegne il calcolatore. Quella che lavora è la seconda routine in linguaggio macchina che va in onda ogni sessantesimo di secondo e suona. Questo esempio può essere provato con qualunque configurazione di memoria.

APPENDICE A

PUNTATORI E REGISTRI DI CONTROLLO

Nel primo K di memoria RAM (byte da 0 a 1023) sono contenute le variabili del sistema. Le coppie di byte che vanno da 43/44 a 65/66 presentano interesse per vedere come viene utilizzata la memoria per un programma Basic. Inoltre il contenuto del byte 648 moltiplicato per 256 dà l'indirizzo di inizio della mappa video. Tale indirizzo può assumere solo due valori:

. 7680 (1E00H) senza espansioni di memoria o con espansione da 3K;
  4096 (1000H) con espansioni superiori a 3K.

Alla mappa video posizionata a 7680 corrisponde la mappa colori a partire da 38400 (9600H), mentre alla mappa video posizionata a 4096 corrisponde la mappa colori a partire da 37888 (9400H).

Il programma PUNTATORI che segue stampa, sul video o sulla stampante, il contenuto dei puntatori citati, il contenuto del byte 648 e i corrispondenti indirizzi di inizio della mappa video e della mappa colori.

```
3000 REM PROGRAMMA PUNTATORI
3001 REM STAMPA CONTENUTO PUNTATORI
3002 DIMD$(11):DATA"INIZ.BASIC","INIZ. VAR.","FINE VAR.","FINE MATR.","IN.STRIN.
"
3003 DATA"FIN.STRIN.","FINE MEM.","LIN.BASIC","LIN.PREC.","PUNT.BASIC"
3004 DATA"LINEA DATA","ARG. DATA"
3005 FORK=0TO11:READD$(K):NEXTK
3006 PRINT"▯RISULTATI SU STAMPANTE":INPUT"S/N";R$
3007 IFR$="N"THEN3011
3008 PRINT"▯▯ACCENDI LA STAMPANTE E POI PREMI UN TASTO"
3009 GETR1$:IFR1$=""THEN3009
3010 OPEN4,4:CMD4
3011 PRINT"▯";:PRINT"IND.PUNT.  CONTENUTO"
3012 K=0
3013 PRINT:FORI=43TO66STEP2:PRINT"▮";I;"▮";I+1;"▮";PEEK(I+1)*256+PEEK(I);:PRINTD
$(K)
```

```
3014 K=K+1:NEXTI
3015 PRINT:PRINT"CONT. BYTE 648=";PEEK(648)
3016 PRINT"INIZ.MAPP.VIDEO ";PEEK(648)*256
3017 X=38400:IFPEEK(648)=16THENX=37888
3018 PRINT"INIZ.MAPP.COLORI ";X
3019 IFR$="S"THENPRINT#4:CLOSE4
3020 STOP
```

Tale programma è stato numerato a partire da 3000 con incremento di 1. Esso può essere inserito in qualunque programma e usato come sottoprogramma (GOSUB 3000), pur di modificare la linea 3020 in: 3020 RETURN.

Alla linea 3002 viene dimensionato il vettore D$ a 11; esso può contenere 12 elementi stringa, partendo con l'indice 0. Seguono 3 istruzioni DATA per le 12 descrizioni; poi alla linea 3005 il vettore viene riempito con le 12 stringhe.
Alla linea 3006 viene chiesto se si desiderano i risultati su video o su stampante, e, nel caso della stampante, viene chiesto di accenderla alla linea 3008. Il controllo viene, nel caso, trasferito da video con: OPEN4,4:CMD4.
I comandi di stampa contengono dei caratteri di controllo validi per il video, che, però, non disturbano la stampante, necessari per far entrare una informazione in 22 caratteri; infatti i numeri alla linea 3013 verrebbero stampati con uno spazio prima e uno spazio dopo e si supererebbe la capacità di una riga video. L'indirizzo contenuto nelle coppie di byte associati per formare un puntatore si ottiene con la solita regola:
256*PEEK(byte pari)+PEEK(byte dispari)
dove il byte dispari precede il byte pari.
Alla linea 3015 viene stampato il contenuto del byte 648, che moltiplicato per 256 dà l'indirizzo d'inizio della mappa video. All'indirizzo della mappa video viene associato il corrispondente indirizzo della mappa colore.

Seguono tre listati ottenuti facendo girare il programma senza espansione di memoria, con espansione da 3K e con espansione da 16K.

```
RISULTATI SU STAMPANTE DEL PROGRAMMA PUNTATORI


VIC-20 SENZA ESPANSIONI DI MEMORIA


     IND.PUNT.  CONTENUTO

      43  44  4097 INIZ.BASIC
      45  46  4760 INIZ. VAR.
      47  48  4788 FINE VAR.
      49  50  4831 FINE MATR.
      51  52  7558 IN.STRIN.
      53  54  7559 FIN.STRIN.
```

```
 55  56  7560 FINE MEM.
 57  58  3013 LIN.BASIC
 59  60  3020 LIN.PREC.
 61  62  4552 PUNT.BASIC
 63  64  3004 LINEA DATA
 65  66  4328 ARG. DATA

CONT. BYTE 648= 30
INIZ.MAPP.VIDEO  7680
INIZ.MAPP.COLORI  38400
```

VIC-20 CON ESPANSIONE 16K

```
IND.PUNT.  CONTENUTO

 43  44  4609 INIZ.BASIC
 45  46  5272 INIZ. VAR.
 47  48  5300 FINE VAR.
 49  50  5343 FINE MATR.
 51  52  24574 IN.STRIN.
 53  54  24575 FIN.STRIN.
 55  56  24576 FINE MEM.
 57  58  3013 LIN.BASIC
 59  60  0 LIN.PREC.
 61  62  5064 PUNT.BASIC
 63  64  3004 LINEA DATA
 65  66  4840 ARG. DATA

CONT. BYTE 648= 16
INIZ.MAPP.VIDEO  4096
INIZ.MAPP.COLORI  37888
```

VIC-20 CON ESPANSIONE DA 3K

```
IND.PUNT.  CONTENUTO

 43  44  1025 INIZ.BASIC
 45  46  1688 INIZ. VAR.
 47  48  1716 FINE VAR.
 49  50  1759 FINE MATR.
 51  52  7678 IN.STRIN.
 53  54  7679 FIN.STRIN.
 55  56  7680 FINE MEM.
 57  58  3013 LIN.BASIC
 59  60  3009 LIN.PREC.
 61  62  1480 PUNT.BASIC
```

```
 63  64  3004 LINEA DATA
 65  66  1256 ARG. DATA

CONT. BYTE 648= 30
INIZ.MAPP.VIDEO  7680
INIZ.MAPP.COLORI  38400
```

Come si vede il programma Basic inizia in posizioni diverse a seconda della memoria RAM presente:

.1025 con espansione da 3;

.4097 senza espansioni;

.4609 con espansioni superiori a 3.

Si ricordi che il calcolatore, sotto Basic, vede una sola espansione di memoria per volta; però, mentre aggiungendo la 3K alla 8K o alla 16K non si hanno sovrapposizioni, aggiungendo la 8K alla 16K si ha la sovrapposizione. Per evitarlo si deve modificare un collegamento all'interno del cartridge.

I byte che vanno da 36864 a 36879 (9000H/900FH) sono i 16 registri del 6561; il loro contenuto è molto importante e va analizzato in binario. Segue il programma REGISTRI che provvede a stampare, sul video o sulla stampante, il contenuto binario dei 16 registri. Tali 16 registri sono memorizzati in una matrice di stringhe (M$(15,7)) in binario e in un vettore (M(15)) di numeri in decimale. Il programma stampa anche i parametri significativi del VIC calcolati usando il contenuto dei 16 registri di controllo.

```
3050 REMSTAMPA REGISTRI CONTROLLO
3051 REM DA 36864 A 36879
3052 DIMM$(15,7),M(15):REM MATRICI PER CONTENUTO BINARIO E DECIMALE 16 REGISTRI
3053 PRINT"█RISULTATI SU STAMPANTE":INPUT"S/N";R$
3054 IFR$="N"THEN3058
3055 PRINT"██ACCENDI LA STAMPANTE  E POI PREMI UN TASTO"
3056 GETR1$:IFR1$=""THEN3056
3057 OPEN4,4:CMD4
3058 PRINT"█";:PRINT"IND.REG.  CONT.BIN."
3059 FORK=0TO15:X=PEEK(36864+K):M(K)=X:PRINT"█";36864+K;
3060 FORI=0TO7:Y=(XAND2↑(7-I))/2↑(7-I):PRINT"█";Y;:X=X-Y*2↑(7-I)
3061 M$(K,I)=MID$(STR$(Y),2,1)
3062 NEXTI:PRINT"█":NEXTK:PRINT:PRINT:
3063 PRINT"SIGNIFICATO CONTENUTO DEI 16 REGISTRI":PRINT
3064 PRINT"DIST. DA SIN. PRIMO CAR. VIDEO ";M(0)AND127
3065 PRINT"DIST. DA ALTO PRIMO CAR. VIDEO ";M(1)
3066 PRINT"NUM. COLONNE MATRICE  VIDEO ";M(2)AND127
3067 PRINT"NUM. RIGHE MATRICE    VIDEO ";(M(3)AND127)/2
3068 A$="8X8":PRINT"MATRICE CARATTERI ";:IFM$(3,7)="1"THENA$="16X8"
3069 PRINTA$;" PUNTI"
3070 PRINT"VALORE RASTER ";:A$="":FORK=0TO7:A$=A$+M$(4,K):NEXTK:A$=A$+M$(3,0)
3071 X=0:FORK=8TO0STEP-1:X=X+VAL(MID$(A$,K+1,1))*2↑(8-K):NEXTK
3072 PRINTX;"   BINARIO=";A$
3073 PRINT"INIZ.MAPPA VIDEO";:X=0:A$="0"
```

```
3074 IFM$(5,0)="0"THENA$="1"
3075 A$=A$+"00"+M$(5,1)+M$(5,2)+M$(5,3)+M$(2,0)
3076 FORK=6TO0STEP-1:X=X+VAL(MID$(A$,K+1,1))*2↑(6-K):NEXTK
3077 PRINTX*512:PRINT"MATRICI CARATTERI";:X=0:A$="0"
3078 IFM$(5,4)="0"THENA$="1"
3079 A$=A$+"00"+M$(5,5)+M$(5,6)+M$(5,7)
3080 FORK=5TO0STEP-1:X=X+VAL(MID$(A$,K+1,1))*2↑(5-K):NEXTK
3081 PRINTX*1024
3082 PRINT"POS.ORIZZ.PENNA LUM.";M(6):PRINT"POS.VERT.PENNA LUM.";M(7)
3083 PRINT"POTENZIOMETRO 1";M(8):PRINT"POTENZIOMETRO 2";M(9)
3084 PRINT"FREQ.PRIMO OSCILLATORE AUDIO";M(10)AND127
3085 PRINT"STATO PRIMO OSCILL.";M$(10,0)
3086 PRINT"FREQ.SEC.OSCILLATORE AUDIO";M(11)AND127
3087 PRINT"STATO SEC.OSCILL.";M$(11,0)
3088 PRINT"FREQ.TERZO OSCILLATORE AUDIO";M(12)AND127
3089 PRINT"STATO TERZO OSCILL.";M$(12,0)
3090 PRINT"FREQ.QUART.OSCILLATORE AUDIO";M(13)AND127
3091 PRINT"STATO QUARTO OSCILL.";M$(13,0)
3092 PRINT"AMPIEZZA SUONO";M(14)AND15
3093 PRINT"COLORE AUSILIARIO MULTICOLOR";M(14)AND240
3094 PRINT"COLORE SFONDO";(M(15)AND240)/16
3095 PRINT"COLORE BORDO";M(15)AND7
3096 PRINT"MODO DIRETTO O INVERSO";(M(15)AND8)/8
3097 IFR$="S"THENPRINT#4:CLOSE4
3098 STOP
```

Il programma è stato numerato a partire da 3050 e anche questo può essere facilmente inserito come sottoprogramma.

Nelle linee da 3059 a 3062 si riempiono il vettore M e la matrice M$, trasformando ogni numero nei bit binari componenti.

Nelle linee da 3063 a 3096 si applicano le regole di utilizzo dei registri per calcolare i parametri significativi. Si fa riferimento alla tabella che segue, dove sono indicati con simboli distinti e numerati i diversi bit componenti i registri e si riporta un riepilogo dei contenuti.

indirizzi		bit							
dec.	esadec.	7	6	5	4	3	2	1	0
36864	9000	I	Sx6	Sx5	Sx4	Sx3	Sx2	Sx1	Sx0
36865	9001	Sy7	Sy6	Sy5	Sy4	Sy3	Sy2	Sy1	Sy0
36866	9002	Mv9	Cv6	Cv5	Cv4	Cv3	Cv2	Cv1	Cv0
36867	9003	R0	Rv5	Rv4	Rv3	Rv2	Rv1	Rv0	D
36868	9004	R8	R7	R6	R5	R4	R3	R2	R1
36869	9005	Mv15	Mv12	Mv11	Mv10	Mc15	Mc12	Mc11	Mc10

36870	9006	Lx7	Lx6	Lx5	Lx4	Lx3	Lx2	Lx1	Lx0
36871	9007	Ly7	Ly6	Ly5	Ly4	Ly3	Ly2	Ly1	Ly0
36872	9008	Px7	Px6	Px5	Px4	Px3	Px2	Px1	Px0
36873	9009	Py7	Py6	Py5	Py4	Py3	Py2	Py1	Py0
36874	900A	S1	F16	F15	F14	F13	F12	F11	F10
36875	900B	S2	F26	F25	F24	F23	F22	F21	F20
36876	900C	S3	F36	F35	F34	F33	F32	F31	F30
36877	900D	S4	F46	F45	F44	F43	F42	F41	F40
36878	900E	Ca3	Ca2	Ca1	Ca0	A3	A2	A1	A0
36879	900F	Cb3	Cb2	Cb1	Cb0	R	Ce2	Ce1	Ce0

CONTENUTI:

I = 1 attiva interlacciamento.

Sx6/Sx0 = distanza della prima colonna di caratteri dal bordo sinistro del video.

Sx7/Sx0 = distanza della prima riga di caratteri dal bordo superiore del video.

Cv6/Cv0 = numero colonne disponibili sul video.

Rv5/Rv0 = numero righe disponibili sul video.

D = selezione matrice caratteri (0 corrisponde a 8x8 e 1 a 16x8).

R8/R1/R0 = contatore raggio di scansione video.

Mv15/Mv10/Mv9 = formazione indirizzo mappa video.

Mc15/Mc10 = formazione indirizzo mappa caratteri.

Lx7/Lx0 = posizione orizzontale light pen.

Ly7/Ly0 = posizione verticale light pen.

F 7/Px0 = valore potenziometro 1.

F 7/Py0 = valore potenziometro 2.

I 16/F10 = frequenza oscillatore audio 1.

S1 = attivazione oscillatore 1.

I 26/F20 = frequenza oscillatore audio 2.

S2 = attivazione oscillatore 2.

I 36/F30 = frequenza oscillatore audio 3.

S3 = attivazione oscillatore 3.

I 46/F40 = frequenza oscillatore audio 4.

S4 = attivazione oscillatore 4.

(a3/Ca0 = colore ausiliario usato in modo multicolor.

3/A0 = volume segnale audio.

b3/Cb0 = colore dello sfondo.

= visualizzazione diretta o inversa.

e2/Ce0 = colore bordo esterno video.

eguono i risultati del programma REGISTRI provato con la configurazione base el VIC e con l'espansione da 16K.

IC-20 SENZA ESPANSIONI DI MEMORIA

```
VIC-20 SENZA ESPANSIONI DI MEMORIA
IND.REG.  CONT.BIN.
 36864  0  0  0  0  1  1  0  0
 36865  0  0  1  0  0  1  1  0
 36866  1  0  0  1  0  1  1  0
 36867  0  0  1  0  1  1  1  0
 36868  1  0  0  1  0  1  1  1
 36869  1  1  1  1  0  0  0  0
```

36870	0	0	0	0	0	0	0	0
36871	0	0	0	0	0	0	0	0
36872	1	1	1	1	1	1	1	1
36873	1	1	1	1	1	1	1	1
36874	0	0	0	0	0	0	0	0
36875	0	0	0	0	0	0	0	0
36876	0	0	0	0	0	0	0	0
36877	0	0	0	0	0	0	0	0
36878	0	0	0	0	0	0	0	0
36879	0	0	0	1	1	0	1	1

SIGNIFICATO CONTENUTO DEI 16 REGISTRI

DIST. DA SIN. PRIMO CAR. VIDEO 12
DIST. DA ALTO PRIMO CAR. VIDEO 38
NUM. COLONNE MATRICE VIDEO 22
NUM. RIGHE MATRICE VIDEO 23
MATRICE CARATTERI 8X8 PUNTI
VALORE RASTER 302 BINARIO=100101110
INIZ.MAPPA VIDEO 7680
MATRICI CARATTERI 32768
POS.ORIZZ.PENNA LUM. 0
POS.VERT.PENNA LUM. 0
POTENZIOMETRO 1 255
POTENZIOMETRO 2 255
FREQ.PRIMO OSCILLATORE AUDIO 0
STATO PRIMO OSCILL.0
FREQ.SEC.OSCILLATORE AUDIO 0
STATO SEC.OSCILL.0
FREQ.TERZO OSCILLATORE AUDIO 0
STATO TERZO OSCILL.0
FREQ.QUART.OSCILLATORE AUDIO 0
STATO QUARTO OSCILL.0
AMPIEZZA SUONO 0
COLORE AUSILIARIO MULTICOLOR 0
COLORE SFONDO 1
COLORE BORDO 3
MODO DIRETTO O INVERSO 1

VIC-20 CON ESPANSIONE 16K

IND.REG.	CONT.BIN.							
36864	0	0	0	0	1	1	0	0
36865	0	0	1	0	0	1	1	0
36866	0	0	0	1	0	1	1	0
36867	1	0	1	0	1	1	1	0
36868	0	0	0	0	0	1	1	0
36869	1	1	0	0	0	0	0	0
36870	0	0	0	0	0	0	0	0
36871	0	0	0	0	0	0	0	0
36872	1	1	1	1	1	1	1	1
36873	1	1	1	1	1	1	1	1
36874	0	0	0	0	0	0	0	0
36875	0	0	0	0	0	0	0	0
36876	0	0	0	0	0	0	0	0
36877	0	0	0	0	0	0	0	0
36878	0	0	0	0	0	0	0	0
36879	0	0	0	1	1	0	1	1

```
SIGNIFICATO CONTENUTO DEI 16 REGISTRI

DIST. DA SIN. PRIMO CAR. VIDEO  12
DIST. DA ALTO PRIMO CAR. VIDEO  38
NUM. COLONNE MATRICE  VIDEO  22
NUM. RIGHE MATRICE    VIDEO  23
MATRICE CARATTERI 8X8 PUNTI
VALORE RASTER  13    BINARIO=00001101
INIZ.MAPPA VIDEO 4096
MATRICI CARATTERI 32768
POS.ORIZZ.PENNA LUM. 0
POS.VERT.PENNA LUM. 0
POTENZIOMETRO 1 255
POTENZIOMETRO 2 255
FREQ.PRIMO OSCILLATORE AUDIO 0
STATO PRIMO OSCILL.0
FREQ.SEC.OSCILLATORE AUDIO 0
STATO SEC.OSCILL.0
FREQ.TERZO OSCILLATORE AUDIO 0
STATO TERZO OSCILL.0
FREQ.QUART.OSCILLATORE AUDIO 0
STATO QUARTO OSCILL.0
AMPIEZZA SUONO 0
COLORE AUSILIARIO MULTICOLOR 0
COLORE SFONDO 1
COLORE BORDO 3
MODO DIRETTO O INVERSO 1
```

Si può provare ad aggiungere al programma la linea 3049, così:

3049 POKE 36867,PEEK(36867)+1

e a farlo girare. La prova può essere fatta solo se si dispone della stampante; infatti sul video si vedranno caratteri senza senso. Si deve rispondere la prima e la seconda volta con RETURN e si vedranno dei risultati corretti sulla stampante, con il bit 0 del byte 36867 a 1, e la matrice caratteri di 16x8 PUNTI.

CARTRIDGE SUPER EXPANDER

Il IC 1211A SUPER EXPANDER CARTRIDGE comprende 4K di ROM che va no a collocarsi all'indirizzo 40960 (A000H), e 3K di RAM che vanno a collocarsi all ndirizzo 1024 (0400H). Il cartridge va montato, come sempre, a calcolatore sp ito; può essere usato anche in presenza delle espansioni da 8K e da 16K, non in pr enza dell'espansione da 3K.

Il ırtridge offre le seguenti prestazioni:

.1 Aggiunge 3K di RAM.
.2 Attiva i tasti funzione.
.3 Aggiunge comandi grafici.
.4 Aumenta le possibilità musicali.
.5 Aggiunge comandi per controllare alcuni registri.

.1 'aggiunta dei 3K di RAM corrisponde a quella che si ottiene con l'espansione R M da 3K. Si ha lo spostamento dell'inizio del programma Basic a 1025, la m pa video rimane a 7680 e la mappa colori a 38400. Usando il comando G APHIC0 o non usando il comando si programma normalmente in Basic con il vi o in condizioni normali.

.2 tasti funzione sono disponibili con delle funzioni preassegnate e precisamente:
F corrisponde a "GRAPHIC"
F corrisponde a "COLOR"
F corrisponde a "DRAW"
F corrisponde a "SOUND"
F corrisponde a "CIRCLE"
F corrisponde a "POINT"
F corrisponde a "PAINT"
F corrisponde a "LIST" + CHR$(13)

Si ricorda che le funzioni pari si ottengono premendo il tasto relativo contemporaneamente o a SHIFT o a COMMODORE. Scrivendo KEY seguito da RETURN compaiono sul video le stringhe di comandi assegnate ai tasti funzione. Queste stringhe possono essere modificate così:

.portando il cursore sulla riga del video relativa alla funzione, modificando il contenuto della stringa e premendo RETURN;
.scrivendo: KEY n,"stringa" e premendo RETURN.
In particolare si possono assegnare stringhe che comprendano anche il RETURN, come è già dall'inizio per la funzione 8. Per esempio, si può assegnare alla funzione 4 il compito di far girare i programmi; si scrive:
KEY 4,"RUN"+CHR$(13) e si preme RETURN. Dopo questa assegnazione per mandare in onda un programma basta premere il tasto funzione 4.

La stringa assegnabile può essere lunga al massimo 128 caratteri e, dal momento che è molto semplice modificare le assegnazioni, si possono preparare delle stringhe lunghe che consentano di stendere più rapidamente un programma, e poi rimodificare le assegnazioni.

.3)Riguardo alla grafica sono possibili 4 situazioni, che si determinano con il comando GRAPHIC seguito da un numero da 0 a 3.
GRAPHIC0 corrisponde allo stato normale del calcolatore. GRAPHIC1 pone il calcolatore nello stato grafico multicolor con il formato dei caratteri 16x8. GRAPHIC2 pone il calcolatore nello stato colore alta risoluzione con il formato dei caratteri 8x8. GRAPHIC3 pone il calcolatore nello stato multicolor o alta risoluzione a seconda del colore che si seleziona per i caratteri: codice maggiore di 7 o minore-uguale a 7. Per controllare i colori sono disponibili 4 registri: sfondo, bordo, caratteri e ausiliario. Al momento dell'accensione i colori disponibili sono ordinatamente: 1, 3, 6 e 0. Ovviamente il quarto registro non viene usato se non si è in multicolor. I comandi per controllare i colori sono:

COLOR sf,bo,car,aus
che agisce su tutti i 4 registri colore;

REGION car
che agisce solo sul registro car.

I numeri dei codici colore sono i soliti. Quando si è in GRAPHIC3 il contenuto del registro "car" determina il modo multicolor o alta risoluzione con la nota regola.

I comandi disponibili sono inoltre: POINT, DRAW, CIRCLE, PAINT, CHAR, SCNCLR. In modo grafico il sistema accetta coordinate x e y, per muoversi sul video, comprese tra 0 e 1023. Sono cioè apparentemente disponibili 1024 punti per ogni dimensione. Infatti la risoluzione del video è a livello di pixel (punto all'interno di una posizione carattere, 8x8 pixel per ogni posizione carattere) e rimane di

17 x184 (22 caratteri x 23 linee), ma vengono accettate coordinate da 0 a 1023 e pr vvede il sistema ad operare le necessarie riduzioni di scala. Queste riduzioni so o fatte con fattore di scala diverso sui due assi, dato che il numero di pixel di ponibili è diverso; il video è rettangolare e non quadrato. Dal punto di vista dei rif rimenti il video si presenta con il punto di coordinate 0,0 nell'angolo in alto a sii stra, l'asse x orientato verso destra e l'asse y orientato verso il basso.

Si iassume la sintassi dei comandi:

P(INT c,x,y
di egna un punto nel colore c in x,y
P(INT c,x1,y1,x2,y2,....yn,yn
di egna n punti del colore c in x1,y1 x2,y2 ...xn,yn
D AW c,x,y TO x1,y1
tr ccia una linea di colore c tra x,y e x1,y1
D AW c,x,y TO x1,y1 TO x2,y2 ...
tr ccia una linea tra x,y e x1,y1, tra x1,y1 e x2,y2, ecc.
D AW c TO x,y
tr ccia una linea dal punto dove si trova (eventualmente 0,0) fino a x,y
C RCLE c,x,y,rx,ry
d egna un cerchio o un ellissi nel colore c, con centro x,y; se rx è circa il 70% di ry si h un cerchio passante per il punto che dista rx e ry dal centro, altrimenti si ha una e ssi
C RCLE c,x,y,rx,ry,ai,af
d egna un arco di cerchio o di ellissi, ai e af rappresentano l'inizio e la fine dell'arco, t iendo conto che 0 corrisponde alla posizione delle ore sul quadrante alle 3, che l' rco si sviluppa in senso orario e che si chiude con il valore 100 ancora sulle 3
F AINT c, x,y
r mpie di colore c la figura all'interno della quale sta il punto x,y; se la figura non è c iusa viene colorato tutto il video; in multicolor il colore deve essere lo stesso del b rdo del disegno
C HAR r,c, "stringa"
s rive a partire dalla riga r e dalla colonna c la stringa
S NCLR
p lisce il video grafico, corrisponde a SHIF/CLR/HOME

I EGION e DRAW non possono essere usati in modo diretto.

S guono 4 programmi esempio che mostrano i diversi effetti provocati dai comandi g afici usandoli nei 3 modi grafici.

```
P OGRAMMA EXP1
1 REM PROVA 3 MODI GRAPHIC
E REM CORRISPONDE AL FORMATO MULTICOLOR
1  GRAPHIC1:CHAR6,9,"GRAPHIC1"
2  GOSUB200:GOSUB300:GOSUB400:GOSUB500
3  REM CORRISPONDE AL MODO ALTA RISOLUZIONE
4  GRAPHIC2:CHAR6,9,"GRAPHIC2"
5  GOSUB200:GOSUB300:GOSUB400:GOSUB500
```

```
68 REM CORRISPONDE AL MODO MULTICOLOR O ALTA RISOLUZIONE
69 REM IN DIPENDENZA DAL CODICE COLORE DEL REGISTRO DEI CARATTERI
70 GRAPHIC3:CHAR6,0,"GRAPHIC3  MULTICOLOR"
74 REM COLORE CHE METTE IL BIT 3 A 1 E QUINDI MULTICOLOR
75 REGION10
80 GOSUB200:GOSUB300:GOSUB400:GOSUB500
84 REM COLORE CHE METTE IL BIT 3 A 0 E QUINDI ALTA RISOLUZIONE
85 REGION0:CHAR6,9,"HI-RES      "
90 GOSUB200:GOSUB300:GOSUB400:GOSUB500
100 END
200 FORK=0TO7:POINT2,K*128,10:NEXTK:RETURN
300 FORK=0TO7:POINT2,K*128,512:NEXTK:RETURN
400 FORK=0TO7:POINT2,K*128,1000:NEXTK:RETURN
500 FORK=1TO1000:NEXTK:RETURN
```

Dimostra l'uso del comando POINT. Si noti la differenza che si ottiene in modo GRAPHIC3 modificando il colore dei caratteri con REGION.

```
PROGRAMMA EXP2

1 REM PROVA LINEE
10 GRAPHIC1:CHAR1,9,"GRAPHIC1"
15 GOSUB100:GOSUB200:GOSUB300:GOSUB400
19 GOSUB500
20 GRAPHIC2:CHAR1,9,"GRAPHIC2"
25 GOSUB100:GOSUB200:GOSUB300:GOSUB400
29 GOSUB500
30 GRAPHIC3:CHAR1,9,"GRAPHIC3":CHAR2,10,"MULTICOLOR"
33 REGION10
35 GOSUB100:GOSUB200:GOSUB300:GOSUB400
36 GOSUB500
37 GRAPHIC3:REGION0:CHAR1,9,"GRAPHIC3":CHAR2,10,"HI-RES"
45 GOSUB100:GOSUB200:GOSUB300:GOSUB400
90 END
100 DRAW2,10,10TO200,200:RETURN
200 DRAW3,50,970TO500,500TO676,850:RETURN
300 DRAW0TO700,924:RETURN
400 FORK=1TO1000:NEXTK:RETURN
500 SCNCLR:GOSUB400:RETURN
```

```
PROGRAMMA EXP3

1 REM PROVA CURVE
10 GRAPHIC1:CHAR1,9,"GRAPHIC1"
15 GOSUB100:GOSUB200:GOSUB300:GOSUB400
19 GOSUB500
20 GRAPHIC2:CHAR1,9,"GRAPHIC2"
25 GOSUB100:GOSUB200:GOSUB300:GOSUB400
29 GOSUB500
30 GRAPHIC3:CHAR1,9,"GRAPHIC3":CHAR2,10,"MULTICOLOR"
33 REGION10
35 GOSUB100:GOSUB200:GOSUB300:GOSUB400
36 GOSUB500
37 GRAPHIC3:REGION0:CHAR1,9,"GRAPHIC3":CHAR2,12,"HI-RES"
45 GOSUB100:GOSUB200:GOSUB300:GOSUB400
90 END
```

```
10  CIRCLE2,512,512,140,200
10  CIRCLE5,512,512,280,400:RETURN
20  CIRCLE2,300,300,100,180:RETURN
30  CIRCLE2,912,870,70,60,0,55:RETURN
40  FORK=1TO1000:NEXTK:RETURN
50  SCNCLR:GOSUB400:RETURN
```

```
PF  GRAMMA EXP4

1   EM PROVA CURVE
1(  GRAPHIC1:CHAR1,9,"GRAPHIC1"
15  GOSUB100:GOSUB200:GOSUB300:GOSUB400
1?  GOSUB500
2(  GRAPHIC2:CHAR1,9,"GRAPHIC2"
25  GOSUB100:GOSUB200:GOSUB300:GOSUB400
2?  GOSUB500
3(  GRAPHIC3:CHAR1,9,"GRAPHIC3":CHAR2,10,"MULTICOLOR"
3?  REGION10
35  GOSUB100:GOSUB200:GOSUB300:GOSUB400
3(  GOSUB500
3?  GRAPHIC3:REGION0:CHAR1,9,"GRAPHIC3":CHAR2,12,"HI-RES"
45  GOSUB100:GOSUB200:GOSUB300:GOSUB400
9(  END
1(  CIRCLE2,512,512,140,200
1(  CIRCLE5,512,512,280,400
1(  PAINT5,712,712:RETURN
2(  DRAW7,10,10TO150,10TO150,150TO10,150TO10,10:PAINT7,20,20:RETURN
3(  CIRCLE7,912,870,70,60:PAINT7,912,870:RETURN
4(  FORK=1TO1000:NEXTK:RETURN
5(  SCNCLR:GOSUB400:RETURN
```

Q esti ultimi dimostrano rispettivamente l'uso di DRAW, CIRCLE e PAINT. T tti i programmi usano CHAR e alcuni anche SCNCLR.
P r tornare al modo normale, quando si è in modo grafico, si può eseguire un c mando GRAFIC0 da programma, oppure premere contemporaneamente RUN-/ TOP e RESTORE.

) Le possibilità musicali aggiunte sono le seguenti:

.) uso di stringhe musicali in una istruzione PRINT;

) uso della tastiera per suonare in modo diretto;

) uso del comando SOUND per suonare accordi.

S no disponibili 5 registri: s1, s2, s3, s4 e s5. I primi 3 corrispondono alle 3 tonalità p ssibili; ogni registro è di un ottava più alta del precedente e dispone di un range di 3 ottave. Il registro s4 serve per creare il rumore bianco e il registro s5 per definire il v lume del suono, tra 0 e 15 (senza usare s5 si può arrivare al massimo al volume 9).

Per usufruire delle possibilità offerte dai punti a e b si deve entrare in MODO MUSICA; questo si ottiene: o premendo contemporaneamente i tasti CTRL e freccia a sinistra, o ponendo i caratteri corrispondenti a questi tasti in una istruzione PRINT subito dopo le virgolette di inizio stringa. In MODO MUSICA il sistema accetta e riconosce solo un gruppo di caratteri e precisamente:

P, Q, V, S, O, T, R per comandi

C, D, E, F, G, A, B per le 7 note musicali

@ per il diesis

$ per il bemolle.

Dopo essere entrati in modo musicale si può suonare in modo immediato usando comandi e note, oppure immettere delle stringhe contenenti comandi e note in istruzioni PRINT all'interno di un programma.
Il significato dei comandi è il seguente:

P abilita la visualizzazione del carattere corrispondente alle note suonate;

Q disabilita l'apparizione del carattere sul video;

Vn dispone il volume a n (da 0 a 9);

Sn seleziona uno tra i tre toni (n=1, 2, 3)

On seleziona una tra le 3 ottave disponibili

Tn determina la durata di una nota nel modo seguente: n può variare da 0 a 9 e le corrispondenti durate sono 4, 6, 8, 12, 16, 24, 32, 64, 128 e 255 sessantesimi di secondo;

R determina una pausa la cui durata dipende dal precedente comando T.

Il comando SOUND consente di suonare accordi; esso si scrive:

SOUND s1, s2, s3, s4, s5 dove

i parametri hanno il significato già visto.

.5 ono disponibili 7 funzioni per leggere il contenuto di particolari registri. Esse sc o: RGR, RCOLR, RDOT, RSND, RPOT, RPEN, RJOY.

R R (n) fornisce il numero del modo grafico nel quale si è. Deve essere scritto un n nero da 0 a 255 entro parentesi, ma questo non influisce sull'operazione. Non p essere usato in modo diretto.

R OLR (n) dove n deve essere da 0 a 3, fornisce il codice colore contenuto nel re itivo registro: 0 per registro sfondo, 1 per bordo, 2 per carattere e 3 per ausiliario.

R OT (x,y) fornisce il colore del punto di coordinate x,y.

R ND (n) fornisce il valore di un registro sonoro. Può essere usata solo dopo S UND; n va da 1 a 5 e si riferisce ai 5 registri sonori.

R OT (n) fornisce il valore di una paddle; n=0 per la prima paddle, n=1 per la se onda. Si ottiene un numero compreso tra 0 e 255 che rappresenta la posizione re ativamente all'angolo alto sinistro del video.

R EN (n) fornisce le coordinate della light-pen; se n=0 dà la x, se n=1 dà la y.

R OY (n) consente di leggere all'interno del calcolatore il movimento o l'azione del jo stick, cosa che è necessaria per programmare giochi. Il comando vuole un n mero n tra parentesi, che però non influenza la lettura; si può usare sempre zero. L risposta ha il seguente significato:

) nessun movimento.
1 movimento verso l'alto
2 movimento verso il basso
4 movimento verso sinistra
5 movimento diagonale sinistra alto
6 movimento diagonale sinistra basso
8 movimento verso destra
9 movimento diagonale destra alto
0 movimento diagonale destra basso
1 8 pressione pulsante

CARTRIDGE VIC STAT

Il CARTRIDGE VIC STAT aggiunge 8K di ROM che vanno a collocarsi dall'indirizzo 40960 all'indirizzo 49151 (A000H-BFFFH); in conseguenza non può essere usato (disponendo del CABINET di espansione) insieme ad altri cartridge, come SUPER EXPANDER, che hanno gli stessi indirizzi ROM. Non si ha aggiunta di RAM e in realtà, data la potenza di programmazione che si aggiunge, sarebbe bene lavorare con una espansione RAM; cosa che rende necessario il cabinet. Dopo averlo inserito, con il calcolatore spento, all'accensione del VIC il programma parte utomaticamente e comincia a girare una parte dimostrativa delle possibilità offerte; si può premere:

F1 per interrompere la dimostrazione e avere disponibile il BASIC + STAT
F3 per interrompere la dimostrazione e avere disponibile solo il BASIC.

Il programma STAT può essere riattivato scrivendo:
SYS 41035 seguito da RETURN;
può essere disattivato scrivendo:
@KILL seguito da RETURN.
Alla partenza il calcolatore si trova nel modo grafico, e in tale modo vengono tracciati i grafici; volendo si può passare al modo testo con PRINT CHR$(14) e ritornare al modo grafico con PRINT CHR$(142).

comandi di STAT iniziano tutti con il carattere "@", così non possono essere confusi con i normali comandi Basic.Sono disponibili 15 comandi che riguardano la grafica, la copia del video e la statistica. I nuovi comandi comportano l'uso di 11 ariabili riservate, che quindi non devono essere usate nei programmi con significato diverso da quello voluto da STAT. Si ricordi che le variabili sono riconosciute in base ai primi 2 caratteri MAX e MA sono la stessa variabile e così pure MIN e MI. e 11 variabili riservate sono:

MV media
A deviazione standard
'A varianza

NP punto di zero
LK gradiente
SK coefficiente di variazione
KK coefficiente di correlazione
BA errore standard
KO correlazione lineare
MI valore minimo
MA valore massimo

dopo l'esecuzione dei comandi che le coinvolgono, in queste variabili si trovano i risultati.

Nei programmi di prova di STAT si deve fare attenzione a non dimensionare matrici di molti elementi, altrimenti sembra che i comandi non funzionano, mentre in realtà manca solo memoria RAM per lavorare.

Si avrà il messaggio ILLEGAL QUANTITY ERROR se non si rispettano i limiti di variabilità dei parametri; SYNTAX ERROR se il comando è scritto male; altri messaggi per eventuali errori di calcolo, come divisioni per zero.
Nei comandi possono essere usati come parametri sia le variabili che le costanti.

COMANDI GRAFICI

Per i comandi di tipo PLOT il video mette a disposizione 2024 punti (44 colonne x 46 righe). Considerando l'asse x in basso al video e l'asse y a sinistra in verticale il campo di variabilità per le coordinate è:
$0 \leqslant x \leqslant 43$ $\quad$ $0 \leqslant y \leqslant 45$

@PLOT,x,y disegna un punto di coordinate x,y

@PLOTD,x,y cancella un punto di coordinate x,y

@PLOTC,x,y controlla la posizione video di coordinate x,y e pone il byte 828 a 1 se il punto è "acceso", a 0 se il punto è "spento"; dopo si deve analizzare il byte 828 leggendolo con la funzione PEEK.

Il comando @PAPER copia sulla stampante il contenuto del video completo; se si vuole copiare solo una parte si deve scrivere così:

@PAPER,TR,BR,LC,RC dove:

TR = riga in alto
BR = riga in basso

LC = colonna a sinistra
RC = colonna a destra

servono a delimitare il quadro.

I tasti di controllo del cursore sono attivi in STAT e quindi è possibile andare a scrivere dove si vuole sul video.

I comandi per tracciare istogrammi sono i seguenti:

@BARV,x1,x2,y traccia una barra verticale con base da x1 a x2 ed altezza y. I limiti sono:
$0 \leqslant x1 \leqslant 20$, $0 \leqslant x2 \leqslant 20$, $0 \leqslant y \leqslant 21$

@BARH,y,x traccia una barra orizzontale alla riga y lunga x; i limiti sono:
$0 \leqslant y \leqslant 17$, $0 \leqslant x \leqslant 19$

Il comando per modificare la scala è:
@SCALE,H,V dove H è l'indice orizzontale e V quello verticale; i limiti sono:
$0 \leqslant H \leqslant 16$, $0 \leqslant V \leqslant 12$

Se non si usa questo comando, sugli assi si va di 5 in 5 e ogni marcatura corrisponde a 1.

Il comando: @LR,G,S,T serve per modificare i colori. G, da 0 a 7, determina il colore dei grafici, S, da 0 a 255, determina il colore dello sfondo e del bordo, T, da 0 a 7, determina il colore del testo. Il comando può essere usato anche solo con il primo parametro o solo con i primi due.

COMANDI PER CALCOLI STATISTICI

Questi comandi lavorano su una matrice che deve essere dimensionata all'inizio del programma e deve avere il nome di una sola lettera; è ammessa una sola dimensione. Per lavorare su dati raggruppati si deve porli nella matrice in modo sequenziale primo/secondo, terzo/quarto, ecc.

@STATP,M,D dove M è il nome della matrice e D è il numero degli elementi della matrice considera i dati accoppiati.

@STATS,M,D dove il significato dei parametri è come sopra ma lavora su dati non raggruppati.

Dopo l'esecuzione di questi comandi si trovano calcolate le variabili:
MV media; VA varianza; SA deviazione standard.

Segue il programma VSTAT1 che usa questi comandi e il listato dei risultati. I dati di calcolo sono generati a caso. Si osservino i diversi risultati con l'uso dei due comandi.

```
1 REM VSTAT1
2 REM STATISTICA
3 REM ALTEZZA/PESO
10 DIM A(100)
15 FORX=1TO100STEP2
20 A(X)=150+10*RND(1)+10*RND(1)+10*RND(1):A(X)=INT(A(X))
25 A(X+1)=50+10*RND(1)+10*RND(1)+10*RND(1):A(X+1)=INT(A(X+1))
30 NEXTX
31 OPEN4,4:CMD4
35 PRINT"         VALORI":PRINT"ALT.  PESO ALT.  PESO":PRINT
40 FORX=1TO100STEP4
45 PRINTA(X);" ";A(X+1);" ";A(X+2);" ";A(X+3)
50 NEXTX
55 @STATP,A,100
56 PRINT:PRINT
57 PRINT"VALORI CALCOLATI CON STATP":PRINT
60 PRINT"MEDIA=";MV
65 PRINT"DEV.STAND.=";SA
70 PRINT"VAR.=";VA
73 PRINT:PRINT
75 PRINT"VALORI CALCOLATI CON STATS":PRINT
79 PRINT"MEDIA=";MV
83 PRINT"DEV.STAND.=";SA
85 PRINT"VAR.=";VA
150 PRINT#4:CLOSE4:STOP
```

Risultati del programma VSTAT1:

```
         VALORI
ALT.  PESO ALT.  PESO

 160   70   169   71
 176   65   173   61
 159   71   170   59
 157   63   166   70
 161   69   162   70
 154   75   173   66
 158   64   169   63
 168   59   167   63
 170   64   166   65
 172   66   158   66
 171   61   155   65
 155   73   162   71
 163   59   164   75
 164   66   161   58
 170   64   166   62
 163   59   168   63
 174   56   167   58
 162   65   173   63
 165   64   171   66
 173   64   162   71
 159   66   167   66
 165   64   158   71
 163   67   172   63
 167   60   170   63
 157   66   176   68


VALORI CALCOLATI CON STATP

MEDIA= 115.28
DEV.STAND.= 50.4047751
VAR.= 2540.64135
```

```
VALORI CALCOLATI CON STATS

MEDIA= 115.28
DEV.STAND.= 50.4047751
VAR.= 2540.64135
```

Il comando:

@LINREG,M,D calcola la regressione lineare e riempie le variabili:
NP punto zero; LK gradiente; SK coeff. variazione; KK coeff. correlazione; BA rrore standard. M è il nome della matrice e D il numero di elementi su cui alcolare.

eguono il programma VSTAT2 che usa questo comando e il listato dei risultati.

```
1 REM VSTAT2
2 REM STATISTICA
3 REM PESO/ALTEZZA
10 DIM A(24)
20 FORX=1TO24:READA(X):NEXTX
31 OPEN4,4:CMD4
35 PRINT"         VALORI":PRINT"PESO  ALT. PESO ALT.":PRINT
40 FORX=1TO24STEP4
45 PRINTA(X);" ";A(X+1);" ";A(X+2);" ";A(X+3)
50 NEXTX
55 @STATP,A,20
60 PRINT:PRINT:PRINT"MEDIA=";MV
65 PRINT"DEV.STAND.=";SA
70 PRINT"VAR.=";VA
80 @LINREG,A,24
85 PRINT"F(X)=";NP;"+(";LK;"*X)"
90 PRINT"]";:PRINT"COEFF.VAR.=";SK
95 PRINT"COEFF.COR.=";KK
100 PRINT"ERR.STAND.=";BA
150 PRINT#4:CLOSE4
190 PRINT"]SCRIVI PESO"
195 PRINT"PER USCIRE 0"
200 INPUT"PESO=";X
205 IFX=0THEN240
210 Y=NP+LK*X
220 PRINT"Y=";Y
230 GOTO200
240 @PAPER:STOP
250 DATA71,160,73,183,64,154,65,168
251 DATA61,159,70,180,65,145,72,210
252 DATA63,132,67,168,64,141,71,200
```

Risultati del programma VSTAT2:

```
        VALORI
PESO  ALT. PESO ALT.

 71   160   73   183
 64   154   65   168
 61   159   70   180
 65   145   72   210
 63   132   67   168
 64   141   71   200


MEDIA= 116.5
DEV.STAND.= 51.5834276
VAR.= 2660.85
F(X)=-133.028791 +( 4.46196711 *X)
COEFF.VAR.= .593136851
COEFF.COR.= .770153784
ERR.STAND.= 15.6641497
```

```
SCRIVI PESO
PER USCIRE 0
PESO=? 70
Y= 179.308907
PESO=? 72
Y= 188.232841
PESO=? 65
Y= 156.999071
PESO=? 67
Y= 165.923006
PESO=? 75
Y= 201.618742
PESO=? 59
Y= 130.227269
PESO=? 0
```

Il comando:

@LINKO,M,D calcola la correlazione lineare e pone il risultato in KO. I parametri sono come sopra.

Il comando:

@MINMAX,M,D trova il minimo e il massimo tra i dati della matrice M e li pone in MI (MIN) e MA (MAX).

Il comando:

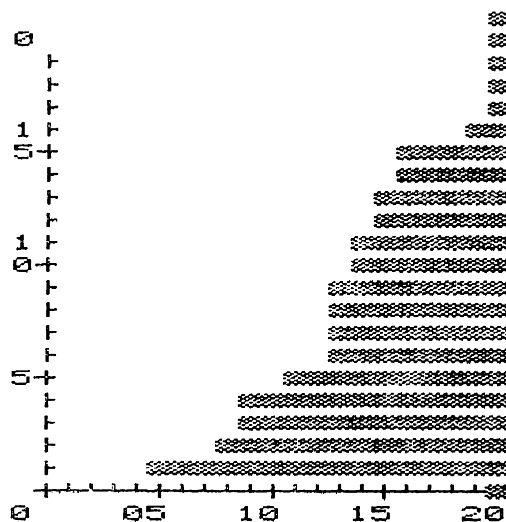
@SO,D ordina in ordine crescente la matrice M, che non viene citata nel comando ed è quella che è stata dimensionata per prima nel programma; lavora su D elementi della matrice.

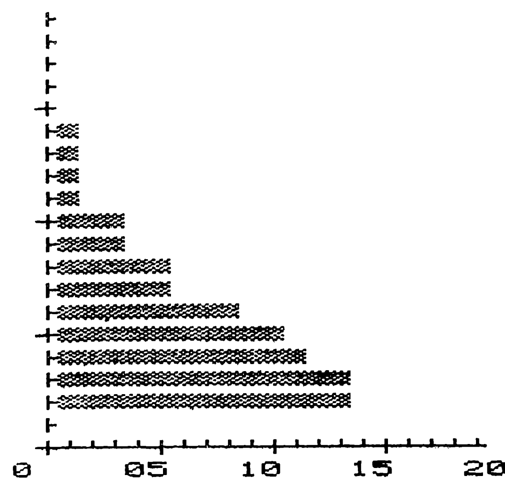
Seguono il programma VSTAT3, che usa i comandi prima elencati, e il listato dei risultati.

```
1 REM VSTAT3
10 DIMA(20):PRINTCHR$(147)
20 PRINT"]RACCOLTA DATI"
30 INPUT"QUANTI DATI: ";N:IFN>20THEN20
40 FORX=1TON:INPUT"DATO:";D
50 A(X)=D:NEXTX:PRINTCHR$(147)
55 OPEN4,4:CMD4
60 PRINT"   ORDINAMENTO"
70 @SO,N
80 K=0:FORX=1TON:PRINTA(X);"    ";:K=K+1:IFK=5THENPRINT:K=0
85 NEXTX
95 @MINMAX,A,N:PRINT"  MINIMO=";MIN
105 PRINT"  MASSIMO:";MAX
110 @STATS,A,N:PRINT"  MEDIA=";MV
120 PRINT"  DEV.STAND.=";SA
130 PRINT"  VAR.=";VA
140 FORX=1TO2500:NEXTX:PRINT"]";
145 PRINT#4:CLOSE4
210 @COLR,0,25
213 Z=MAX-MIN
215 FORX=1TON
217 B=INT(A(X)/Z*21):IFB>21THENB=21
220 @BARV,X,X,B
230 @COLR,2
233 IFX/2=0THEN235
234 @COLR,6
235 NEXTX
240 @PAPER
250 FORX=1TO2500:NEXTX:PRINT"]";
254 IFN>17THENN=17
255 FORX=1TON
260 B=INT(A(X)/Z*19):IFB>19THENB=19
265 @BARH,X,B
270 NEXTX
275 @PAPER
300 END
```

Risultati del programma VSTAT3:

```
  ORDINAMENTO
12      34      40      45      65
90      97      123      234      234
345      345      543      678      765
876      876      876      987      1236
 MINIMO= 12
 MASSIMO: 1236
 MEDIA= 425.05
 DEV.STAND.= 392.00906
 VAR.= 153671.103
```





Si noti la differenza negli istogrammi tracciati con i due comandi disponibili.

APPENDICE D

CARTRIDGE VIC GRAF

I CARTRIDGE VIC GRAF serve per studiare le funzioni. Si installa, come al s lito a calcolatore spento e parte automaticamente all'accensione. Si tratta di un p ogramma chiuso, nel senso che svolge le sue funzioni, ma non può essere i tegrato in un altro programma. Il menù del programma è il seguente:

1=PLOT THE GRAPH	traccia il grafico
2=NEW FUNCTION	nuova funzione
3=AXIS INTERSECT	intersezione con gli assi
4=EXPOSE FUNCTION	scrive la funzione
5=FIND MAX E MIN	trova massimo e minimo
6=CORR.IN EQUAT.	correzione equazione
7=EVAL.INTEGRAL	calcolo integrale
TOP RETURN MENU	stop per tornare al menù

me si vede non esiste un comando per tornare al BASIC; quando si è terminato si ve spegnere il calcolatore e togliere la ROM.

e varie opzioni sono attivate usando i tasti funzione che si trovano sulla destra lla tastiera; per ottenere le funzioni pari si deve premere insieme il tasto SHIFT.

lla partenza, il messaggio DEFIN FUNCTION consente di scrivere la funzione le può occupare fino a 76 caratteri, si può scrivere e correggere al solito modo; uando si preme RETURN la funzione viene accettata. Se si è dimenticato di rreggere qualcosa lo si può fare con l'opzione F6. In qualunque momento il mando STOP fa tornare al MENU' e da lì si può ripartire con una nuova scelta.

opo l'opzione F1 vengono chiesti gli estremi dell'intervallo (STATE RAN-E:(X0, XN)=?) e poi viene tracciato il grafico. Dopo il grafico appare in alto la manda: NEW RANGE? YES NO; se si risponde Y vengono chiesti di nuovo gli tremi dell'intervallo per tracciare un nuovo grafico. Questo consente di ristudiare funzione con maggiore dettaglio in una parte di intervallo. Se si risponde N si ha

la domanda: WANT GRAPHICAL DATA? (desideri dati relativi al grafico?). Alle domande che terminano con punto interrogativo si deve rispondere o con Y (per si) o con N (per no). Alle domande di dati si deve rispondere con i dati richiesti. Altre domande del colloquio sono:

WANT AXIS INTERSECT?	vuoi il valore dell'intersezione con gli assi
STATE AN PROX.VALUE	stabilisci un valore approssimato (usa il metodo di NEWTON-RAPHSON)
INTERSECT X=....	intersezione x=...
WANT EXTREMAL POINT?	vuoi i punti estremanti
GIVE RANG OF SEARCH	intervallo per la ricerca
WANT AN INTEGRAL?	vuoi calcolare l'integrale
RANGE OF INTEGR=?	intervallo per l'integrazione

I grafici vengono costruiti calcolando 160 punti, che vengono mostrati sul video.

APPENDICE E

CARTRIDGE PROGRAMMER'S AID

Questo cartridge è molto utile sia per i principianti che per i programmatori esperti. Esso, se si possiede il cabinet di espansione, può essere usato insieme ad espansioni di memoria RAM e anche insieme a VICMON e a SUPER EXPANDER. Come sempre, inserire il cartridge con il VIC spento.
Per attivarlo scrivere SYS28681, per escluderlo KILL seguito da RETURN. Tutti i comandi vengono accettati quando si preme RETURN.
Il sistema, con incorporato il cartridge, si può trovare in due stati: PROGRAM MODE e EDIT MODE. In ognuno di questi due stati i tasti funzione rispondono a funzioni diverse. Dopo l'attivazione si è in PROGRAM MODE; si passa in EDIT MODE semplicemente scrivendo: EDIT e RETURN. Analogamente si torna in PROGRAM MODE scrivendo: PROG seguito da RETURN. Un altro metodo per passare da un modo all'altro è premere contemporaneamente CTRL e F1. Si elencano le funzioni assegnate ai tasti Fx nei due modi, e si fa notare che in alcuni casi nella funzione è già incorporato il "RETURN" e la "aperta parentesi".

PROGRAM MODE		EDIT MODE	
F1	LIST	F1	LIST
F2	MID$(	F2	AUTO
F3	RUN RETURN	F3	RUN RETURN
F4	LEFT$(	F4	DELETE
F5	GOTO	F5	FIND
F6	RIGHT$(	F6	CHANGE
F7	INPUT	F7	TRACE RETURN
F8	CHR$(	F8	STEP RETURN
F9	EDIT RETURN	F9	PROG RETURN
F10	GOSUB	F10	RENUMBER
F11	RETURN RETURN	F11	MERGE
F12	STR$(	F12	OFF RETURN

Per ottenere le 12 funzioni con i tasti si deve tener presente che:

.premendo semplicemente i tasti si ottiene F1, F3, F5 e F7;
.premendo contemporaneamente il tasto SHIFT si ottiene F2, F4, F6 e F8;
.premendo contemporaneamente il tasto CTRL si ottiene F9, F10, F11 e F12.

Se si usa KILL per uscire dal programma, i tasti funzione mantengono le assegnazioni ricevute.
Le stringhe assegnate ai tasti funzione possono essere modificate; si deve fare attenzione, perchè cambiando modo si ritorna all'assegnazione iniziale del modo selezionato.
Tutti i comandi sono disponibili nei due modi, ma se non sono assegnati ai tasti funzione, essi devono essere scritti digitandoli.
In modo EDIT sono disponibili funzioni speciali che si ottengono premendo insieme a CTRL alcune lettere; esse sono:
CTRL A per ottenere lo scrolling verso l'alto
CTRL E per potersi muovere con il cursore tra le virgolette senza lasciare traccia
CTRL L per cancellare tutti i caratteri della riga dopo il cursore
CTRL N per cancellare tutto il programma dopo il cursore
CTRL Q per ottenere lo scrolling verso il basso
CTRL U per cancellare sul video la linea dove sta il cursore

Questi comandi agiscono sul video, per ottenere l'effetto anche in memoria si deve usare il RETURN (o altre modalità richieste) per confermare la modifica; essi possono essere molto utili in fase correzione o modifica programmi.

Si elencano i comandi disponibili.

AUTO oppure AUTO inizio, intervallo
servono per ottenere la numerazione automatica delle linee mentre si scrive un programma. Nel primo caso il numero linea parte da 100 e si incrementa di 10. Premendo il RETURN dopo il numero di linea che appare viene disabilitato il comando.

RENUMBER inizio, incremento
serve per rinumerare un programma partendo da un numero di linea e usando un determinato incremento. Agisce anche sui GOTO e GOSUB. La rinumerazione avviene dall'inizio del programma.

DELETE pr.num.-ult.num.
num.
-num.
num.-

s ve per cancellare linee di programma comprese tra due numeri di linea, una sola li ea, dall'inizio fino a una linea, da una linea in poi.

F ND codice Basic, range
stringa, range
carattere, range

s rve per ricercare nel range indicato, cioè tra due numeri di linea, fino a una certa l iea, da una linea in avanti, o in tutto il programma (se manca il range) il soggetto i dicato. Vengono mostrate sul video tutte le linee di programma dove compare l ırgomento della ricerca. Per rallentare un eventuale scrolling si può tenere premu- ' ' il tasto CTRL.

HANGE vecchio, nuovo, range
:rve per sostituire a "vecchio" "nuovo" nel range indicato. Se manca il range la cerca è su tutto il programma. "vecchio" e "nuovo" possono essere un codice o na stringa.

.EY num., stringa oppure KEY
ɜnza parametri dà la lista delle stringhe assegnate ai tasti funzione. Con i parametri ɜrve per cambiare la stringa assegnata a un tasto funzione, quello di cui si cita il umero. La stringa assegnata può esssere al massimo di 10 caratteri. Per aggiungere RETURN alla stringa si deve scrivere +CHR$(13).

HELP
ʒerve per mostrare sul video la linea dove si è verificato un errore durante l'esecuzio- ıe di un programma. Se si è premuto STOP, la linea mostrata è l'ultima eseguita.

DUMP
ista le variabili del programma nell'ordine nel quale sono state definite e i loro :ontenuti. Non vengono listate le variabili con indice.

TRACE
crea una finestra sul video nella quale vengono mostrati i numeri delle linee eseguite dal programma. Per rallentare la velocità si può premere o SHIFT o CTRL.

STEP
fa eseguire il programma una istruzione per volta, mostrando il numero della

istruzione che eseguirà subito dopo in una finestra sul video; per proseguire si deve premere o SHIFT o CTRL. Se si premono contemporaneamente i due tasti il programma prosegue con continuità.

OFF
serve per annullare l'effetto o di TRACE o di STEP.

MERGE nome, dn
serve per fondere con il programma presente in memoria il programma "nome" memorizzato sulla periferica dn. Se i due programmi che si fondono hanno alcuni numeri di linea doppi, vengono mantenute le doppie linee e bisogna andare a correggerle.

Questo cartridge è molto utile sia in fase di creazione di programmi, che in fase di messa a punto degli stessi. In fase di scrittura programma si può risparmiare tempo e fatica assegnando alcune stringhe che si pensa possano ripetersi spesso, ai tasti funzione e richiamandole con un solo tasto; si può inoltre usare la numerazione automatica. In fase di prova si può usare TRACE, STEP e HELP, DUMP e OFF. In fase di correzione servono FIND, DELETE, CHANGE, MERGE e RENUMBER.

APPENDICE F

CARTRIDGE TURTLE GRAPHICS

Questo cartridge è molto adatto per mettere in contatto con il calcolatore le persone assolutamente inesperte e i bambini a partire dai sei anni. Si tratta di un facsimile della parte grafica del famoso linguaggio LOGO ideato da Seymour Papert al MIT negli USA.
Il programma si sviluppa per mezzo di semplici menù che appaiono sul video; basta seguire il menù e in breve ci si accorge che si riesce a dominare il calcolatore. Il nome TURTLE (tartaruga) si ispira al LOGO; qui in realtà non si vede una tartaruga, ma un semplice asterisco che ha come base di partenza l'angolo in alto a sinistra del video. Questo asterisco può essere spostato sul video lasciando una traccia o non lasciandola; quindi si possono realizzare disegni a piacere, usando come carattere tracciatore uno dei caratteri del VIC. Naturalmente restano disponibili le caratteristiche di colore e suono proprie del VIC.
Quasi senza accorgersene, si imparano i concetti fondamentali della programmazione, compresi i cicli e i salti condizionati. I programmi che si scrivono possono essere memorizzati sia su nastro che su disco e poi richiamati da altri programmi per creare facilmente disegni sempre più complicati. Il cartridge dispone di un buon numero di comandi, ma lavorando si possono creare delle routine che vengono poi memorizzate con un nome su cassetta o su disco, e formano una serie di nuovi comandi in aggiunta a quelli del cartridge. Si può scrivere un semplice programma che disegna una casetta sul video, si memorizza con il nome "casa", si richiama assegnando dimensioni diverse agli elementi e subito nasce una casa diversa dalla precedente.

Il menù principale è il seguente:

Add lines	aggiunge linee di programma
Insert lines	inserisce linee di programma
Delete lines	cancella linee di programma
Replace lines	sostituisce linee di programma
List program	lista il programma
Print program	stampa il programma
Save to tape or disk	memorizza su nastro o disco

Get off tape or disk	legge da nastro o disco
Execute a program	esegue un programma
Trace a program	mostra i passi di programma

i comandi entrano scrivendo la prima lettera e vengono poste domande semplici per condurre ad eseguirli.

I comandi disponibili per scrivere un programma sono:

PEN DOWN	penna giù, consente di disegnare
TURTLE COLOR	per scegliere il colore
CHARACTER TO	per scegliere il caratt. tracciatore
MOVE TO	per spostarsi a riga e colonna
RIGHT	per andare a destra
DOWN	per andare in giù
LEFT	per andare a sinistra
UP	per andare in sù
STOP	per fermare il programma
PEN UP	per alzare la penna (non si scrive)
SCREEN COLOR	scelta colore video
BORDER COLOR	scelta colore bordo
CALCULATE	per assegnare valori o calcolare
LABEL	per assegnare nomi
USE	per usare una routine
SET COLUMN	per definire posizione colonna
SET ROW	per definire posizione riga
WAIT	per creare una pausa
BEEP	per emettere un suono
NO BEEP	per fermare un suono
LOOP	per iniziare un ciclo
LOOP END	per chiudere un ciclo
FORWARD INDEX	per incrementare un indice di controllo
TURN RIGHT	per girare di 90 gradi a destra
TURN LEF	per girare di 90 gradi a sinistra
TURN AROUND	per cambiare direzione
ROUTINE END	fine routine
JUMP	per saltare
IF	per operare scelte
REMARK	per inserire annotazioni
CLEAR SCREEN	per pulire il video
PEN ERASE	per azzerare dove passa
PEN REVERSE	per operare in colori invertiti

TEXT	per scrivere caratteri
SET DELAY TO	per modificare la velocità
CHECK FOR SYMBOL	per ricercare un simbolo

Come si vede da questa rapida carrellata sono presenti tutti i comandi che consentono di scrivere programmi completi. La stesura dei programmi è semplificata dal fatto che si mette una istruzione per riga, da una sintassi semplice (uno spazio tra gli elementi), dal facile inserimento o cancellazione di righe con rinumerazione automatica.

APPENDICE G

CARTRIDGE WORDCRAFT

WORDCRAFT 20 è un cartridge, prodotto e distribuito dalla AUDIOGENIC SOFTWARE, che contiene un programma di wordprocessing (elaborazione di testi); esso consente di scrivere e modificare sul video testi, lettere, documenti, elenchi, etc., in modo da produrli nella forma finale desiderata, e infine stamparli.
In sostanza è una raffinatissima macchina da scrivere elettrica, con una quantità di automatismi, ma soprattutto con una memoria di notevole capacità che mantiene al suo interno il testo sotto elaborazione; la stampa del testo viene fatta solo alla fine, quando la sua forma è perfetta.
I testi vengono elaborati nella memoria del VIC, e possono essere conservati su nastro e su disco.
Le istruzioni di comando di WORDCRAFT sono di due tipi: i "comandi" che producono un effetto globale su tutto il testo, e i "controlli", che possono avere validità limitata ad una parte di testo indicata dall'utente.
La pagina di start-up, che compare sul video quando è inserito il cartridge WORDCRAFT, consente di selezionare il tipo di stampante, il dispositivo di memoria di massa che si intende usare, la quantità di memoria che si intende utilizzare sotto BASIC. Definiti questi preliminari, si lancia WORDCRAFT con il tasto F1.
La pagina standard di WORDCRAFT contiene 25 linee di 24 caratteri ognuna; le prime 3 linee sono riservate a informazioni di sistema e alle istruzioni di comando; sulle altre compare il testo.

I comandi di WORDCRAFT sono elencati brevemente di seguito:

. n
cancella il testo presente in memoria

. s, nome
salva il testo in memoria su nastro o su disco, con il nome specificato

. r, nome
sosituisce con il testo in memoria il documento già registrato, su nastro o su disco, con il nome specificato

. g, nome
carica in memoria, da disco o da nastro, il testo che ha il nome specificato

. $
fa comparire sul video la directory del disco

. v, numero unità
riordina e compatta il contenuto del disco, rendendo disponibile il maggior spazio possibile

. c
definisce come memoria di massa il registratore a cassette

. d
definisce come memoria di massa il disco; se ce n'è più di uno va specificato il "numero di unità"

. w, nn
fissa il numero "nn" di caratteri della riga di testo (da 3 a 99)

. l, nn
fissa il numero di linee "nn" della pagina di testo (da 1 a 60)

. b
elimina il "bip" prodotto dalla pressione dei tasti

. e, parola d'ordine
registra i testi sotto una parola d'ordine, al massimo di 16 caratteri, nota solo all'utente

. e
annulla il comando precedente

. p, pagine, tipo, copie
produce la stampa delle pagine specificate, con le parti evidenziate sottolineate o in neretto, a seconda del tipo, e nel numero di copie specificato, del testo che si trova in memoria

. a, numeri
invia alla stampante i numeri (con valore compreso tra 0 e 254) che rappresentano i valori ASCII dei caratteri che si vogliono stampare

. “, nn, tipo di carta
fissa la lunghezza fisica della pagina della stampante, e definisce il tipo di carta usata

. j,n o j,y
attiva o annulla la “giustificazione” del testo sul margine destro

. f, nome, pagine
inserisce, in punti prefissati del testo in memoria, le pagine del documento che ha il nome specificato

. m, nome, pagine
aggiunge al testo presente in memoria le pagine del testo che ha il nome specificato

Per dare a WORDCRAFT un'istruzione di controllo, occorre premere prima il tasto CBM; le principali istruzioni di controllo sono elencate brevemente di seguito.

. RETURN
inizio di una nuova linea

. +n
salto di n linee

. HOME
inizio di una nuova pagina

. DEL
cancella la parte di testo a partire dal cursore fino alla fine della linea

. e
cancella una parte di testo

. INST
apre uno spazio nel testo per l'inserimento di una parte nuova

. m
sposta parti di testo da un punto ad un altro

. r
come la precedente, ma lascia la parte di testo anche nella posizione originaria

. =
centra una linea di testo

. #
passa in modo "ruler" (righello), in cui si possono definire margini e tabulazioni con il tasto "freccia in alto" premuto nella posizione desiderata

. t
posiziona il cursore sulla tabulazione successiva

.
allinea i punti decimali

. [
fa rientrare una parte di testo alla tabulazione successiva

.]
annulla il comando precedente

. ( )
definiscono una parte di testo che, in fase di stampa, sarà sottolineata o in neretto, a seconda del tipo scelto nel comando "p".

. s
ricerca una o più parole nel testo, a partire dalla posizione in cui si trova il cursore

. x o z
associato al comando precedente, sostituisce alla frase trovata un'altra

. ?
indica i punti del testo in memoria in cui si vogliono inserire parti prelevate da un altro testo

. c
annulla tutte le istruzioni di controllo presenti in quella linea

. n tipo
annulla l'istruzione di controllo specificata con "tipo"

Esistono poi alcune altre istruzioni di controllo che consentono spostamenti rapidi del cursore sul testo; inoltre alcune delle istruzioni di controllo precedenti possono essere date premendo uno dei tasti di funzione a destra sulla tastiera.
Un'applicazione particolarmente utile di WORDCRAFT è la personalizzazione di documenti; è possibile ad esempio scrivere una volta sola il testo di una lettera circolare, e personalizzarlo automaticamente in un secondo tempo, inserendo in ogni lettera l'indirizzo di un destinatario diverso. Oppure è possibile completare un testo base con paragrafi particolari, prelevati da un file.

SET CARATTERI

TABELLA CODICI VIC MODO CURSOR UP

D3	D2	D1	D0	D7	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
				D6	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
				D5	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
				D4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
D3	D2	D1	D0	HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0		POS	SP	ø	@	P					SP				SP	
0	0	0	1	1		CURSOR DOWN	!	1	A	Q				CURSOR UP						
0	0	1	0	2		SVS ON	"	2	B	R				REV OFF						
0	0	1	1	3			#	3	C	S										
0	1	0	0	4			$	4	D	T										
0	1	0	1	5			%	5	E	U										
0	1	1	0	6			&	6	F	V										
0	1	1	1	7			'	7	G	W										
1	0	0	0	8	BS		(	8	H	X										
1	0	0	1	9			)	9	I	Y										
1	0	1	0	A	NL	SUB	*	:	J	Z										
1	0	1	1	B		ESC	+	;	K	[										
1	1	0	0	C			,	<	L	¥										
1	1	0	1	D	CR		—	=	N	]										
1	1	1	0	E	SO		.	>	O	↑		π						π		
1	1	1	1	F	SI		/	?	P	←										π

TABELLA CODICI VIC MODO CURSOR DOWN

				D7	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
				D6	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
				D5	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
				D4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
D3	D2	D1	D.0	HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0		POS	SP	Ø	@	p		P			SP			P	SP	
0	0	0	1	1		CURSOR DOWN	!	1	a	q	A	Q		CURSOR UP			A	Q		
0	0	1	0	2		SVS ON	"	2	b	r	B	R		REV OFF			B	R		
0	0	1	1	3			#	3	c	s	C	S					C	S		
0	1	0	0	4			$	4	d	t	D	T					D	T		
0	1	0	1	5			%	5	e	u	E	U					E	U		
0	1	1	0	6			&	6	f	v	F	V					F	V		
0	1	1	1	7			'	7	g	w	G	W					G	W		
1	0	0	0	8	BS		(	8	h	x	H	X					H	X		
1	0	0	1	9			)	9	i	y	I	Y					I	Y		
1	0	1	0	A	NL	SUB	*	:	j	z	J	Z					J	Z		
1	0	1	1	B		ESC	+	;	k	[	K						K			
1	1	0	0	C			,	<	l	¥	L						L			
1	1	0	1	D	CR		–	=	m	]	M						M			
1	1	1	0	E	SO		.	>	n	↑	N						N			
1	1	1	1	F	SI		/	?	o	←	O						O			

TABELLA CONVERSIONE ESADECIMALE/DECIMALE

DEC.	ESAD.	DEC.	ESAD.	DEC.	ESAD.	DEC.	ESAD.
0	0	64	40	128	80	192	C0
1	1	65	41	129	81	193	C1
2	2	66	42	130	82	194	C2
3	3	67	43	131	83	195	C3
4	4	68	44	132	84	196	C4
5	5	69	45	133	85	197	C5
6	6	70	46	134	86	198	C6
7	7	71	47	135	87	199	C7
8	8	72	48	136	88	200	C8
9	9	73	49	137	89	201	C9
10	A	74	4A	138	8A	202	CA
11	B	75	4B	139	8B	203	CB
12	C	76	4C	140	8C	204	CC
13	D	77	4D	141	8D	205	CD
14	E	78	4E	142	8E	206	CE
15	F	79	4F	143	8F	207	CF
16	10	80	50	144	90	208	D0
17	11	81	51	145	91	209	D1
18	12	82	52	146	92	210	D2
19	13	83	53	147	93	211	D3
20	14	84	54	148	94	212	D4
21	15	85	55	149	95	213	D5
22	16	86	56	150	96	214	D6
23	17	87	57	151	97	215	D7
24	18	88	58	152	98	216	D8
25	19	89	59	153	99	217	D9
26	1A	90	5A	154	9A	218	DA
27	1B	91	5B	155	9B	219	DB

DEC.	ESAD.	DEC.	ESAD.	DEC.	ESAD.	DEC.	ESAD.
28	1C	92	5C	156	9C	220	DC
29	1D	93	5D	157	9D	221	DD
30	1E	94	5E	158	9E	222	DE
31	1F	95	5F	159	9F	223	DF
32	20	96	60	160	A0	224	E0
33	21	97	61	161	A1	225	E1
34	22	98	62	162	A2	226	E2
35	23	99	63	163	A3	227	E3
36	24	100	64	164	A4	228	E4
37	25	101	65	165	A5	229	E5
38	26	102	66	166	A6	230	E6
39	27	103	67	167	A7	231	E7
40	28	104	68	168	A8	232	E8
41	29	105	69	169	A9	233	E9
42	2A	105	6A	170	AA	234	EA
43	2B	106	6B	171	AB	235	EB
44	2C	107	6C	172	AC	236	EC
45	2D	108	6D	173	AD	237	ED
46	2E	109	6E	174	AE	238	EE
47	2F	110	6F	175	AF	239	EF
48	30	111	70	176	B0	240	F0
49	31	112	71	177	B1	241	F1
50	32	113	72	178	B2	242	F2
51	33	114	73	179	B3	243	F3
52	34	115	74	180	B4	244	F4
53	35	116	75	181	B5	245	F5
54	36	117	76	182	B6	246	F6
55	37	118	77	183	B7	247	F7
56	38	119	78	184	B8	248	F8
57	39	120	79	185	B9	249	F9
58	3A	121	7A	186	BA	250	FA
59	3B	122	7B	187	BB	251	FB
60	3C	123	7C	188	BC	252	FC
61	3D	124	7D	189	BD	253	FD
62	3E	125	7E	190	BE	254	FE
63	3F	126	7F	191	BF	255	FF

L. 22.000 Cod. 338D ISBN 88-7056-139-9

Il libro si pone l'obiettivo di soddisfare diverse esigenze. I capitoli che trattano i file su disco e cassetta, la stampante VIC 1515, alcuni cartridge come VIC STAT, VIC GRAF, SUPER EXPANDER etc, rappresentano una perfetta integrazione al primo volume "IMPARIAMO A PROGRAMMARE IN BASIC CON IL VIC/CBM", e si rivolgono prevalentemente agli utenti BASIC del VIC.
Per chi invece desidera approfondire anche l'aspetto hardware, per arrivare ad una programmazione più sofisticata, questo libro propone alcuni capitoli di grande interesse sulle porte di I/O, sul chip di interfaccia video (VIC 6561) e sul linguaggio macchina del calcolatore.



GRUPPO
EDITORIALE
JACKSON

Rita Bonelli
Daria Gianni

ALLA SCOPERTA DEL VIC 20

architettura e tecniche di programmazione

89